International Journal of Approximate Reasoning 138 (2021) 38–66

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

A compositional approach to probabilistic knowledge compilation

Giso H. Dal [a,*], Alfons W. Laarman [b], Arjen Hommersom [a,c], Peter J.F. Lucas [a,d]

[a] Institute for Computing and Information Sciences, Radboud University, the Netherlands
[b] Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
[c] Faculty of Science, Open University, the Netherlands
[d] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, the Netherlands

ARTICLE INFO

Article history:
Received 19 July 2020
Received in revised form 29 June 2021
Accepted 12 July 2021
Available online 2 August 2021

Keywords:
Bayesian inference
Knowledge compilation
Partitioning
Model counting

ABSTRACT

Bayesian networks (BN) are a popular representation for reasoning under uncertainty. The analysis of many real-world use cases, that in principle can be modeled by BNs, suffers however from the computational complexity of inference. Inference methods based on Weighted Model Counting (WMC) reduce the cost of inference by exploiting patterns exhibited by the probabilities associated with BN nodes. However, these methods require a computationally intensive compilation step in search of these patterns, which effectively prohibits the handling of larger BNs. In this paper, we propose a solution to this problem by extending WMC methods with a framework called *Compositional Weighted Model Counting* (CWMC). CWMC reduces compilation cost by partitioning a BN into a set of subproblems, thereby scaling the application of state-of-the-art innovations in WMC to scenarios where inference cost could previously not be amortized over compilation cost. The framework supports various target representations that are less or equally succinct as decision-DNNF. At the same time, its inference time complexity $\mathcal{O}(n \exp(w))$, where n is the number of variables and w is the tree-width, is comparable to mainstream algorithms based on variable elimination, clustering and conditioning.

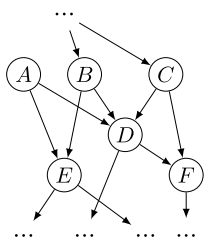
1. Introduction

The field of probabilistic inference has made considerable progress in the past three decades with the development of novel probabilistic graphical models, such as Bayesian networks (BNs) and chain graphs [10]. In particular, Bayesian networks (BNs) have become popular graphical models for reasoning under uncertainty, with applications in areas such as medical diagnosis, speech recognition, weather forecasting, data mining, and so on [28,41]. Despite progress in probabilistic inference, its NP-hardness remains a stumbling block for using exact inference. Often, researchers resort to employing approximate probabilistic reasoning under such circumstances, which could provide practical results, but the computational complexity of approximations remains NP-hard [11]. The current article focuses on exact approaches.

Unfortunately, BNs are not able to take advantage of many forms of independence that could improve their conciseness even further. These forms include causal independence [31], context-specific independence (CSI) [4], and determinism [22].

* Corresponding author.
E-mail addresses: gdal@cs.ru.nl (G.H. Dal), a.w.laarman@liacs.leidenuniv.nl (A.W. Laarman), arjen.hommersom@ou.nl (A. Hommersom), peter.lucas@utwente.nl (P.J.F. Lucas).

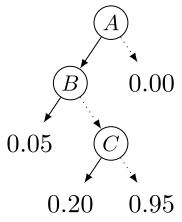
https://doi.org/10.1016/j.ijar.2021.07.007



(a) Bayesian network.

A	B	C	$P(D \mid A,B,C)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0.95
1	0	1	0.20
1	1	0	0.05
1	1	1	0.05

(b) D's CPT.



(c) Decision tree.

Fig. 1. Bayesian network with context-specific independence and determinism.

These are collectively referred to as *local structure*. Fig. 1 shows a partial BN in which local structure is not exploited. CSI exists when probabilities in a conditional probability table (CPT) show uniformity for multiple configurations of the variables they depend on. Note that determinism means that probabilities in a CPT are equal 0 or 1. Consider the CPT in Fig. 1b and recall that $P(D=1 \mid A,B,C) = 1 - P(D=0 \mid A,B,C)$, for any value of A, B, and C. It shows that any conditional probability of $D=1$ given evidence $A=0$ is equal to zero. Fig. 1c shows how to represent the same CPT as a decision tree using 4 probabilities instead of 8. Similarly, causal dependence can be captured concisely using decision diagrams [45, Th.3].

Knowledge compilation (KC) [18] is a probabilistic reasoning approach that avoids the limitations of BNs by 'compiling' propositional theories into a symbolic representation –like the decision tree and diagrams above– that can be queried efficiently. As the compilation step can be done offline, it can spend considerable computational resources, as long as it yields a small symbolic representation. The queries can then be performed online, using for example *weighted model counting* [6], which only takes linear time in the size of the symbolic representation [18]. Various symbolic formalisms have been proposed for KC, e.g., [34],[29] and [44], which all have different characteristics in terms of compilation effort and query times. And because this symbolic approach represents local structure concisely, like the decision tree in Fig. 1c and the diagrams [45, Th.3], KC is generally seen as a method that renders many practical reasoning problems tractable [33].

Although the cost of KC can be amortized over multiple inference queries, it has still proven too costly in many scenarios [1,8,14,33,35]. In particular, the compilation step might fail due to a lack of resources, limiting the applicability of KC. *The current article proposes* **Compositional Weighted Model Counting (CWMC)**, a framework for probabilistic inference that partitions the compilation into subproblems, which are recomposed in the inference query. It builds upon, and extends [15]. This compositional approach offers various advantages. In the first place, it can reduce the effort spent on compilation because the decomposition of a propositional theory is known to yield smaller symbolic representations, which has previously been shown in model checking [25,39,42] and is confirmed by our experiments (which show orders of magnitudes improvement in Table 3). This improvement is offset by a potential increase in the time spent on inference, although our experiments still demonstrate good performance due to the smaller representations. In the second place, the compositional approach is independent of the chosen symbolic target representation, as we demonstrate by using four different target representations in our experiments. As a consequence, the approach is to a certain extent orthogonal to the exploitation of local structure by those representations as local structure can still be exploited within the partitioned subproblems. For instance, we show that causal dependence is fully exploited when using decision diagram representations in the partitions.

CWMC consists of four phases that are outlined in Fig. 2. It also illustrates a comparison to traditional WMC. CWMC employs a divide-and-conquer strategy where probabilistic networks are partitioned into independent subproblems. Each subproblem forms a component that is compiled into a symbolic representation. The compiled subproblems are then composed, circumventing any further compilation cost, such that probabilistic inference can be performed. An algorithm is provided to evaluate the composed representation in order to perform inference with partitioned probability spaces.

This paper is organized as follows. Preliminaries are provided (Section 2). Then, we discuss the theoretical basis in CWMC (Section 3) and the algorithm that exploits these ideas (Section 4). We show that this algorithm has a complexity described by $\mathcal{O}(n \exp(w))$ (Section 5). Methods are provided for finding partitionings and compilation orders to support CWMC (Section 6). Symbolic representations are supported that are less or equally succinct as decision-DNNF, e.g., SDDs, OBDDs, WPBDDs (Section 7). Finally, empirical evaluation shows that both compilation and inference cost are reduced while employing CWMC, sometimes by multiple orders of magnitude (Section 8).

2. Preliminaries and background

For convenience, we summarize some basic set and graph theory and also go through the basics of Bayesian networks. We follow [30] in our notation. More detail will be provided about methods underlying weighted model counting, as this forms the foundation on which the rest of the paper is built.

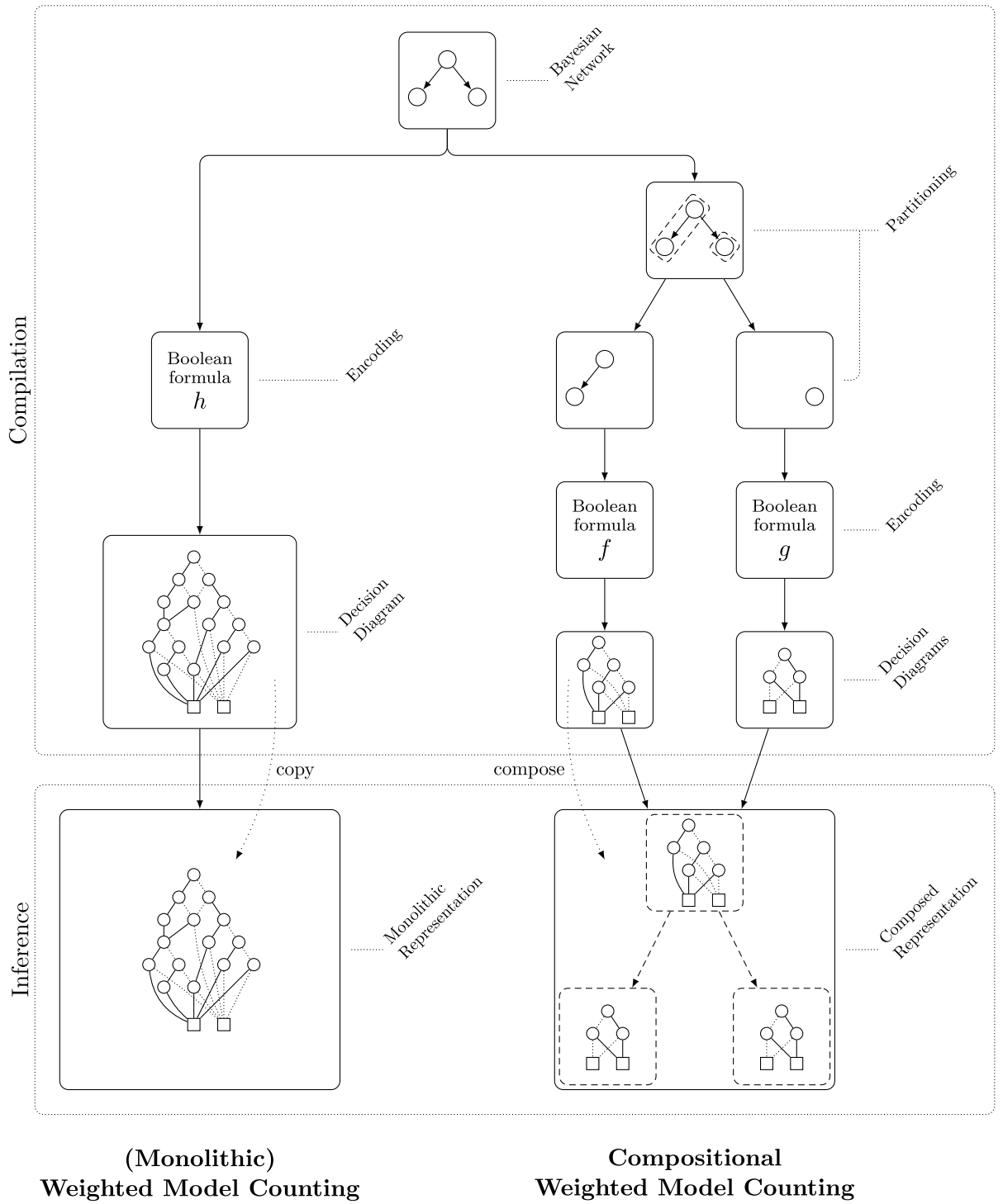


Fig. 2. The compositional framework.

2.1. Set and graph theory

The reader is assumed to be familiar with basic set theory. We use $\subset$ for the strict (proper) subset relation, and $\subseteq$ for the non-strict (improper) subset relation. The symbol $\emptyset$ denotes the empty set, and $|X|$ denotes the cardinality of set X, i.e., its number of elements.

Definition 1 *(Partial ordering).* Let X be a set and $\circ \subseteq X \times X$ a binary relation, and $(X, \circ)$ an *ordered set*. Then $a,b \in X$ are said to be *comparable* if $a \circ b$ or $b \circ a$. Relation $\circ$ is a *weak strict partial order* if:

1. If $a \circ b$ and $b \circ c$ then $a \circ c$ (transitivity), and
2. If $a \circ b$ then not $b \circ a$ (asymmetry),
3. Not $a \circ b$ nor $b \circ a$ does not imply $a = b$ (weakness).

Definition 2 *(Total ordering).* The set $(X, \circ)$ is called *totally ordered* if every pair of elements in set X is comparable by $\circ$. Otherwise, it is *partially ordered*.

We use $\prec$ to denote a *strict* binary relation. Within the context of this article, we only use partial orders that are both weak and strict, and will henceforth simply refer to them as partial orders.

A (directed) graph $\mathcal{G}$ is defined as a pair (V, E), where V is a set of numbers, called *nodes*, and *edges* $E \subseteq V \times V$ are ordered pairs of nodes. An undirected graph $\mathcal{G}$ is a graph with a symmetric edge relation, i.e., $(u, v) \in E$ iff $(v, u) \in E$. Graph $\mathcal{G}_A = (A, E_A)$ is an *induced subgraph* of $\mathcal{G}$ if $A \subseteq V$ and $E_A \subseteq E \cap (A \times A)$, and $\mathcal{G}_A$ is *induced* by A if $E_A = E \cap (A \times A)$. The *undirected version* $\mathcal{G}^{\sim}$ of $\mathcal{G}$ is induced by V that additionally has an edge (v, u) for each $(u, v) \in E$. We further define the following functions for nodes $A \subseteq V$ and $v \in V$:

$$\text{(children)} \quad \mathsf{ch}(A) \triangleq \{\mathsf{ch}(v) \mid v \in A\} \setminus A, \quad \text{with } \mathsf{ch}(v) \triangleq \{u \in V \mid (v, u) \in E\}, \tag{1}$$

$$\text{(parents)} \quad \mathsf{pa}(A) \triangleq \left\{\mathsf{pa}(v) \mid v \in A\right\} \setminus A, \quad \text{with } \mathsf{pa}(v) \triangleq \{u \in V \mid (u, v) \in E\}, \tag{2}$$

$$\text{(family)} \quad \mathsf{fa}(A) \triangleq \{\mathsf{fa}(v) \mid v \in A\}, \quad \text{with } \mathsf{fa}(v) \triangleq \{v\} \cup \mathsf{pa}(v), \tag{3}$$

$$\text{(ancestors)} \quad \mathsf{an}(v) \triangleq \text{ the smallest } A \subseteq V \text{ s.t. } \mathsf{pa}(v) \subseteq A \text{ and } \mathsf{pa}(A) \subseteq A \tag{4}$$

$$\text{(descendants)} \quad \mathsf{de}(v) \triangleq \text{ the smallest } A \subseteq V \text{ s.t. } \mathsf{ch}(v) \subseteq A \text{ and } \mathsf{ch}(A) \subseteq A \tag{5}$$

$$\text{(descendants \& } v\text{)} \quad \mathsf{su}(v) \triangleq \mathsf{de}(v) \cup \{v\} \tag{6}$$

$$\text{(root)} \quad \mathsf{rt}_{\mathcal{G}} \triangleq \left\{v \in V \mid \mathsf{pa}_{\mathcal{G}}(v) = \emptyset\right\}_1 \tag{7}$$

For directed graphs, we visually represent nodes by circles and edges by arrows; for undirected graphs, nodes are also represented by circles and edges as lines. We further allow nodes $v \in V$ to be labeled with a variable; the associated variable is denoted X_v. The set of variables associated with nodes $A \subseteq V$ is denoted by X_A, e.g., the variables associated with parents $\mathsf{pa}(v)$ of v are denoted $X_{\mathsf{pa}(v)}$, and the variable associated with the i^{th} parent is denoted by $X_{\mathsf{pa}_i(v)}$. If necessary, the relevant graph $\mathcal{G}$ is indicated, e.g., $X_{\mathsf{pa}_{\mathcal{G}}(v)}$.

A *path* of length n from node u to v is a sequence of distinct nodes $u = s_0, \ldots, s_n = v$ such that $(s_{i-1}, s_i) \in E$ for all $i = 1, \ldots, n$. A graph is *cyclic* if it contains a path where begin and end points coincide; otherwise it is *acyclic*. Graph $\mathcal{G}$ is a Directed Acyclic Graph (DAG) if it is directed and acyclic. Graph $\mathcal{G}$ is *connected* if there is at least one path in $\mathcal{G}^{\sim}$ between every pair of nodes. A graph $\mathcal{G}$ can be decomposed into the union of connected components, where a *connected component* is an A-induced subgraph $\mathcal{G}_A$ of $\mathcal{G}$ that is connected, and there is no edge in $\mathcal{G}$ between A and $V \setminus A$. A *component* can consist of multiple connected components.

Graph $\mathcal{G}$ is a *tree* if it is a DAG and each node has exactly one incoming edge, except for one: the root, which has zero incoming edges.

2.2. Bayesian networks

A Bayesian network (BN) is a probabilistic graphical model (PGM) representing a concise factorization of a discrete probability distribution, by modeling conditional independence relations among variables. *Factoring* consists here of writing probabilities explicitly as conditional on other variables, represented as parent nodes in a BN (omitting, as much as possible, any conditional independencies). This factorization is usually smaller or simpler than the original. BNs reduce the size of representing a probability distribution from $\mathcal{O}(2^n)$ to $\mathcal{O}(n2^m)$, where n is the number of variables and m is the maximum number of parents of any node.

Let $X = \{X_1, \ldots, X_n\}$ be a set of random variables, that is sometimes interpreted as a n-tuple $X = (X_1, \ldots, X_n)$. We make no distinction between singleton sets $X = \{X_1\}$ and the variable X_1. Values of a variable X_1 are denoted in lower case, e.g., $x_1 \in \mathsf{val}(X_1)$. We denote with $P(X = x)$ the probability that $(X_1, \ldots, X_n) = (x_1, \ldots, x_n)$, i.e. $X_i = x_i$, for $i = 1, \ldots, n$. Let $J \subseteq \mathbb{N}$, $I \subseteq J$, be (finite) sets of indices, $X = \{X_i \mid i \in J\}$, then $X_I = \{X_i \mid i \in I, X_i \in X\}$.

Definition 3 *(Bayesian Networks).* A *Bayesian network* $\mathcal{B} = (\mathcal{G}, P)$ is a DAG $\mathcal{G} = (V, E)$, with nodes V and edges $E \subseteq V \times V$, that models a factorization of joint probability distribution $P(X_V)$ defined over random variables X_V as:

$$P(X_V = x_V) = \prod_{v \in V} P(X_v = x_v \mid X_{\mathsf{pa}(v)} = x_{\mathsf{pa}(v)}), \tag{8}$$

A	B	$P(X)$
1	1	0.4
1	2	0.4
2	1	0.05
2	2	0.05
3	1	0
3	2	0.1

(a) Full joint distribution.

$$P(X) = P(B|A)P(A)$$

(b) Bayesian network.

$P(A=1)$	$P(A=2)$	$P(A=3)$
0.8	0.1	0.1

A	$P(B=1\|A)$	$P(B=2\|A)$
1	0.5	0.5
2	0.5	0.5
3	0	1

(c) Conditional probability tables.

Fig. 3. Bayesian network with local structure.

such that there is a one-to-one correspondence between nodes V and variables X_V, and the conditional probability distribution of $X_v \in X_V$ given its parents $X_{\mathsf{pa}(v)}$ is specified as $P(X_v \mid X_{\mathsf{pa}(v)})$.

Definition 4 *(Cardinality).* Let X be a set of variables. The dimension, or *cardinality*, of $X_i \in X$ is denoted $\mathsf{car}(X_i) \triangleq |\mathsf{val}(X_i)|$, where $\mathsf{val}(X_i)$ denotes the set of X_i's values. Furthermore, car determines the cardinality of $Y \subseteq X$:

$$\mathsf{car}(Y) \triangleq \prod_{Y_i \in Y} \mathsf{car}(Y_i). \tag{9}$$

Example 1 illustrates the graphical representation of a BN, alongside its factorization, and conditional distributions as *Conditional Probability Tables* (CPTs). Note that this simple example is designed to demonstrate the techniques presented throughout this paper, and should not be considered a typical Bayesian network.

Example 1 *(Bayesian Network).* Fig. 3 shows a BN $\mathcal{B}$ defined over variables $X = \{A, B\}$ (Fig. 3b), its CPTs (Fig. 3c) and corresponding full joint probability distribution (Fig. 3a).

The CPTs of BN $\mathcal{B}$ exhibit local structure in the form of context-specific independence and determinism: probabilities show uniformity (equality) given multiple configurations, and 0 and 1 probabilities are present. It is therefore possible to find a more concise representation when taking this into account.

Let X be a set of random variables, $A, B \subseteq X$, then the posterior probability distribution of B with *evidence* $A = a$ is $P(B \mid A = a)$. This distribution is obtained from P after probabilistic updating, in a process called *inference*. This key problem in Bayesian reasoning is addressed in this article.

Let $\mathbb{B}$ and $\mathbb{R}$ denote the Boolean and real domain, respectively. We use potential functions $\varphi : \mathbb{B}^m \longrightarrow \mathbb{R}_0^+$, as generalizations of probability distributions. Although a potential function is not necessarily a probability distribution, we can use them to represent any conditional probability distribution by defining:

$$\varphi(x_1, \ldots, x_m) \triangleq P(X_1 = x_1, \ldots, X_i = x_i \mid X_{i+1} = x_{i+1}, \ldots, X_m = x_m)$$

Given two potential functions φ and ψ defined on sets of variables X_V and X_W respectively, we define the *product* of φ and ψ, as:

$$(\varphi \times \psi)(X_{V \cup W}) \triangleq \varphi(X_V) \cdot \psi(X_W)$$

Furthermore, potential functions can be marginalized. Suppose $S \subseteq V$, then $\sum_S \varphi$ is also a potential defined by:

$$\left(\sum_S \varphi\right)(V \setminus S) \triangleq \sum_S \varphi(V)$$

2.3. Inference by weighted model counting

The WMC approach attempts to improve on the computational advantages of the factorization of a BN by employing additional algebraic manipulations. The process is typically done in a framework where a BN is encoded as a Boolean formula [6]. This formula is then *compiled* to a symbolic representation, e.g., a normal form, that respects the factorization and improves it further to allow for more efficient inference. The computational complexity of inference is linear in the size of this compiled symbolic representation if it adheres to a set of key properties identified by Darwiche [18]. We now review this method and its 3 main steps in some detail, after we reiterate some basic definitions of Boolean logic. This section ends with a discussion of the decisions diagram variant used in this article.

2.3.1. Boolean logic

A *literal* l is a Boolean variable x, or its negation $\bar{x}$. We use $\mathbf{1}$ to denote the true and $\mathbf{0}$ to denote the false values. A *propositional formula* (proposition for short) ψ is a literal, Boolean value ($\mathbf{0}$ or $\mathbf{1}$), or a composite proposition with connectives such as negation $\overline{\varphi}$, conjunction ($\varphi \wedge \psi$), disjunction ($\varphi \vee \psi$), and implication ($\varphi \implies \psi$), where φ and ψ are propositions, with precedence of the connectives in that (descending) order. A proposition φ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals ($l_1 \vee \cdots \vee l_n$). If whenever φ is satisfied by a *model* M (a truth assignment to Boolean variables), the proposition ψ will also be satisfied by M, this will be denoted by $\varphi \models \psi$.

A proposition can be represented as a Boolean function $f : \mathbb{B}^m \to \mathbb{B}$ defined over m Boolean variables. The *conditioning* of f on x_i is defined as the projection $f_{|x_i \leftarrow b}(x_1, \ldots, x_i, \ldots, x_m) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_m)$, with $b \in \{\mathbf{0}, \mathbf{1}\}$. We use $f_{|x_i}$ and $f_{|\overline{x_i}}$ to denote $f_{|x_i \leftarrow 1}$ and $f_{|x_i \leftarrow 0}$, respectively. A Boolean function f *depends* on x_i if $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \neq f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$. In the following we will often not make a distinction between a proposition φ and its associated Boolean function f.

2.3.2. Boolean encoding, compilation and inference

Below, the three steps in turning a Bayesian network into a logical representation that is suitable for efficient weighted model counting are reviewed.

Step 1: Bayesian network encoding BNs are defined over multi-valued domains and an encoding is required to transition to the Boolean domain. To achieve this, we use a *one-hot encoding* for the variables, which introduces a Boolean variable x_i for each unique variable-value pair $v \in \mathsf{val}(X_i)$ [14].

Definition 5 *(Bayesian Network encoding).* Let $\mathcal{B} = (\mathcal{G}, P)$ be a BN, with $\mathcal{G} = (V, E)$. The *encoding* of variables X_V is equal to a set of Boolean atoms $\mathsf{at}(X_V)$:

$$\mathsf{at}(X_V) \triangleq \bigcup_{X_v \in X_V} \mathsf{at}(X_v), \tag{10}$$

$$\text{where } \mathsf{at}(X_v) \triangleq \bigcup_{u \in \mathsf{val}(X_v)} \mathsf{at}(u), \tag{11}$$

where $\mathsf{at}(u)$ is a Boolean atom x, $\mathsf{at}(X_v) = \{x_1, \ldots, x_n\}$ has a one-to-one mapping to values $\mathsf{val}(X_v) = \{u_1, \ldots, u_n\}$, and for $\mathsf{at}(X_V)$ holds that $\mathsf{at}(X_k) \cap \mathsf{at}(X_l) = \emptyset$ for all $X_k, X_l \in X_V$ with $X_k \neq X_l$. Semantically, we say that X_v is equal to its j^{th} value if $x_j = \mathbf{1}$.

Furthermore, we define $\mathsf{bnvar}(x_j) = X_v$ and $\mathsf{bnval}(x_j) = u_j$, with $x_j \in \mathsf{at}(X_v), \mathsf{at}(u_j) = x_j, u_j \in \mathsf{val}(X_v)$ and $X_v \in X_V$. Finally, the function $\mathsf{li}(X_V)$ provides the *literals* $\{x, \overline{x} \mid x \in \mathsf{at}(X_V)\}$, i.e., atoms and their negations.

Probabilities are encoded in a *propositional knowledge base* according to Definition 6.

Definition 6 *(Knowledge base).* A *propositional probabilistic knowledge base* (PPKB) is a set of weighted formulas $\{\langle\varphi_1, \omega_1\rangle, \ldots, \langle\varphi_n, \omega_n\rangle\}$, where each propositional formula φ_i is associated with *weight atom* ω_i, symbolically representing probability $\mathsf{pr}(\omega_i) \in [0, 1]$. The function pr returns 1 if no probability is specified for a given literal; in particular, $\mathsf{pr}(\overline{\omega_i}) \triangleq 1$. A PPKB is written as a proposition by conjoining each pair $\langle\varphi_i, \omega_i\rangle$, that is syntactic sugar for $(\varphi_i \implies \omega_i)$, or clause $(\overline{\varphi_i} \vee \omega_i)$.

The factorization of BN $\mathcal{B} = (\mathcal{G}, P)$, with $\mathcal{G} = (V, E)$, is encoded by adding clauses that ensure consistency with network instantiations, i.e., assignments to the probabilistic variables, and clauses that capture the relation between instantiations and probabilities. A weighted clause is added to the PPKB for every probability in conditional distribution $P(X_v \mid X_{\mathsf{pa}(v)})$:

$$\bigcup_{v \in V} \left(\overbrace{\left\langle \bigwedge_{y \in \mathsf{at}(X_v)} \overline{y}, \boldsymbol{\omega_0} \right\rangle}^{\text{at least one}} \cup \overbrace{\left(\bigcup_{y,z \in \mathsf{at}(X_v),\ y \neq z} \langle y \wedge z, \boldsymbol{\omega_0} \rangle \right)}^{\text{at most one}} \cup \overbrace{\bigcup_{y \in \mathsf{at}(X_v),\ z_1,\ldots,z_m \in \prod_{u \in \mathsf{pa}(v)} \mathsf{at}(X_u)} \left\langle y \wedge z_1 \wedge \cdots \wedge z_m, \boldsymbol{\omega}(y, z_1, \ldots, z_m) \right\rangle}^{\text{probability encoding}} \right),$$

where $\boldsymbol{\omega}(y, z_1, .., z_m)$ is a fresh atom for the probability $P(X_v = \text{bnval}(y) \mid \text{bnvar}(z_1) = \text{bnval}(z_1), \ldots, \text{bnvar}(z_m) = \text{bnval}(z_m))$ and $\boldsymbol{\omega_0}$ an atom for the probability $\mathbf{0}$.[1] This interpretation will later be used during inference. The encoding introduces a unique atom, symbolizing the probability weight, for every unique probability in the CPT of X_v, thereby allowing multiple equivalent probabilities to be associated with the same atom.

Example 2 *(Bayesian Network encoding).* Let BN $\mathcal{B} = (\mathcal{G}, P)$, with $\mathcal{G} = (V, E)$, be defined over variables $X_V = \{A, B\}$ as described in Example 1 and shown in Fig. 3. To encode the BN we add Boolean variables $\text{at}(A) = \{a_1, a_2, a_3\}$ and $\text{at}(B) = \{b_1, b_2\}$ (Definition 5).

PPKB KB is constructed by adding for every $X_i \in X$ one-hot constraints for the values of X_i based on $\text{at}(X_i)$ ('at least one' and 'at most one'), and adding a weighted formula for every probability in X_i's CPT (the 'probability encoding'). The probability mapping for variable A is $\text{pr}(\omega_1) = 0.8$, $\text{pr}(\omega_2) = 0.1$ and for variable B is $\text{pr}(\omega_3) = 0.5$. We do not encode deterministic probabilities per CPT. Instead, $\text{pr}(\boldsymbol{\omega_0}) = 0.0$ and $\text{pr}(\boldsymbol{\omega_1}) = 1.0$, and pr returns 1.0 in all other cases.

The part of the PPKB for variable A is as follows:

$$\left\{ \overbrace{\langle \overline{a_1} \wedge \overline{a_2} \wedge \overline{a_3}, \boldsymbol{\omega_0} \rangle}^{\text{at least one}}, \overbrace{\langle a_1 \wedge a_2, \boldsymbol{\omega_0} \rangle, \langle a_1 \wedge a_3, \boldsymbol{\omega_0} \rangle, \langle a_2 \wedge a_3, \boldsymbol{\omega_0} \rangle}^{\text{at most one}}, \overbrace{\langle a_1, \omega_1 \rangle, \langle a_2, \omega_2 \rangle, \langle a_3, \omega_2 \rangle, \ldots}^{\text{probability encoding}} \right\}$$

Weighted clauses are created for variable B analogously. The CNF representation follows directly from the PPKB definition:

$$(a_1 \vee a_2 \vee a_3 \vee \boldsymbol{\omega_0}) \wedge (\overline{a_1} \vee \overline{a_2} \vee \boldsymbol{\omega_0}) \wedge (\overline{a_1} \vee \overline{a_3} \vee \boldsymbol{\omega_0}) \wedge (\overline{a_2} \vee \overline{a_3} \vee \boldsymbol{\omega_0}) \wedge$$
$$(\overline{a_1} \vee \omega_1) \wedge (\overline{a_2} \vee \omega_2) \wedge (\overline{a_3} \vee \omega_2) \wedge (b_1 \vee b_2 \vee \boldsymbol{\omega_0}) \wedge (\overline{b_1} \vee \overline{b_2} \vee \boldsymbol{\omega_0}) \wedge$$
$$(\overline{a_1} \vee \overline{b_1} \vee \omega_3) \wedge (\overline{a_1} \vee \overline{b_2} \vee \omega_3) \wedge (\overline{a_2} \vee \overline{b_1} \vee \omega_3) \wedge (\overline{a_2} \vee \overline{b_2} \vee \omega_3) \wedge (\overline{a_3} \vee \overline{b_1} \vee \boldsymbol{\omega_0}) \wedge (\overline{a_3} \vee \overline{b_2} \vee \boldsymbol{\omega_1})$$

The combination of the clauses $(b_1 \vee b_2 \vee \boldsymbol{\omega_0})$ and $(\overline{b_1} \vee \overline{b_2} \vee \boldsymbol{\omega_0})$ act as the constraints to ensure that variable B can only be assigned one value. Either b_1 or b_2 can be true, not both, semantically imply $B = 1$ or $B = 2$. Note that clauses with deterministic weights can be simplified, e.g., $(b_1 \vee b_2 \vee \boldsymbol{\omega_0}) \equiv (b_1 \vee b_2 \vee \mathbf{0}) \equiv (b_1 \vee b_2)$, which will further simplify the encoding.

The encoding includes the following models for variable A:

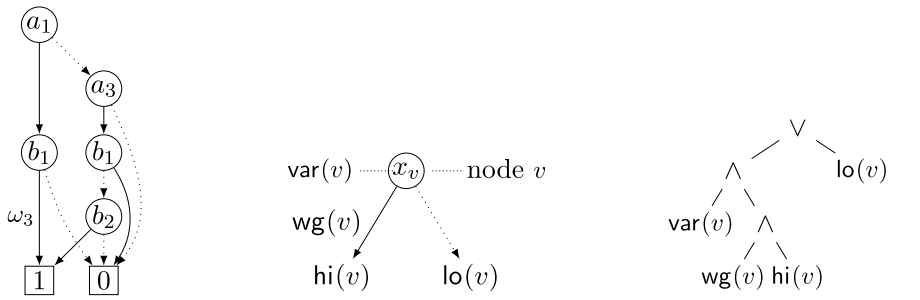
	Models					Weights
1	a_1	$\overline{a_2}$	$\overline{a_3}$	ω_1	$\overline{\omega_2}$	$\text{pr}(\omega_1) \cdot \text{pr}(\overline{\omega_2}) = 0.8 \cdot 1 = 0.8$
2	$\overline{a_1}$	a_2	$\overline{a_3}$	$\overline{\omega_1}$	ω_2	$\text{pr}(\overline{\omega_1}) \cdot \text{pr}(\omega_2) = 1 \cdot 0.1 = 0.1$
3	$\overline{a_1}$	$\overline{a_2}$	a_3	$\overline{\omega_1}$	ω_2	$\text{pr}(\overline{\omega_1}) \cdot \text{pr}(\omega_2) = 1 \cdot 0.1 = 0.1$

Note that the weighted model count sums to 1.0 for this selection of models. However, there are more models, e.g., model a_1,$\overline{a_2}$,$\overline{a_3}$,ω_1,ω_2, model $\overline{a_1}$,a_2,$\overline{a_3}$,ω_1,ω_2, etc. Only minimal models sum to 1.0, i.e., models with the most amount of negations.

To yield a valid probability distribution we require that only minimal models participate in the model count. Minimal models are those with the most amount of negations. One way to remove non-minimal models is by extending the encoding. For every encoding formula $\langle y \wedge z_1 \wedge \cdots \wedge z_m, \boldsymbol{\omega}(y, z_1, \ldots, z_m) \rangle$ we add $\boldsymbol{\omega}(y, z_1, \ldots, z_m) \implies (y \wedge z_1 \wedge \cdots \wedge z_m)$. This is also referred to as a *completion* formula [23]. For example, the completion formulae for variable A in Example 2 are $(a_1 \vee \overline{\omega_1})$, $(a_2 \vee \overline{\omega_2})$, and $(a_3 \vee \overline{\omega_3})$. In our actual implementation, completion formulae are not explicitly added. We discuss how non-minimal models are handled in Section 2.5.

Step 2: compiling to a concise symbolic representation To refactor a joint probability distribution $P(X)$, a concise symbolic representation is obtained through compilation that represents $P(X)$'s encoding. This seems an indirect route, but using decision diagrams such as OBDD [5], d-DNNF [7], WPBDD [14], etc., to represent the PPKB provides access to a rich set of tools specifically developed for the efficient manipulation of Boolean functions. These representations all take a subset of a functionally complete Boolean language, a normal form, condensing the corresponding syntax trees into DAGs. Any Boolean function can be represented within the chosen subset, but the conciseness differs, e.g., there are functions with exponentially sized OBDD representation, but polynomial d-DNNF representations [18]. Manipulations, on the other hand, such as conjoining or negating two represented formulas, are not always efficient. For example, conjunction is efficient in OBDD, but not in d-DNNF representations. Indeed, every representation has a different balance between conciseness and manipulation costs [18], so the selection must be made wisely.

[1] So we have $\text{pr}(\boldsymbol{\omega_0}) \triangleq \mathbf{0}$ and $\text{pr}(\boldsymbol{\omega}(y, z_1, \ldots, z_m)) \triangleq P(X_v = \text{bnval}(y) \mid \text{bnval}(z_1), \ldots, \text{bnval}(z_m))$.



(a) WPBDD example. **(b)** WPBDD node. **(c)** WPBDD node circuit.

Fig. 4. The semantics of a WPBDD node.

Step 3: inference Representations used in the context of WMC typically offer linear time inference. More specifically, this holds for representations that are decomposable, deterministic and smooth [18]. OBDD, d-DNNF, WPBDD, and many more have these properties. To realize model counting, and thereby inference, each node in the representation is visited once while computing its semantics, hence the complexity of inference is linear. We provide the semantics for one such representation, i.e., WPBDDs, in Section 2.4.

2.4. Decision diagrams

Many variants and extensions of decision diagrams exist for representing (and manipulating) Boolean functions [5,7, 18]. A *Weighted Positive Binary Decision diagram* (WPBDD) is one such decision diagram that is particularly well suited to model probability distributions encoded as Boolean formulas [14]. A WPBDD is an ordered BDD that represents a concise factorization of a Boolean formula f as a (rooted) directed acyclic graph with decision nodes, and two terminal nodes labeled with 1 and 0 (Fig. 4a). Each non-terminal node v is labeled with a Boolean variable $\mathsf{var}(v) = x_v$ and has two children, $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$, with a set of weight variables $\mathsf{wg}(v)$ at the edge to node $\mathsf{hi}(v)$. Edges to nodes $\mathsf{hi}(v)$ and $\mathsf{lo}(v)$ are solid and dotted, respectively, as shown in Fig. 4b. Its logical equivalent is shown in Fig. 4c. The subgraph induced by nodes $\mathsf{su}(v)$ represents Boolean function f_v. If v is the root of the WPBDD, then $f_v = f$. Formula f_v is created by conditioning f on the variables by which nodes U on the path from the root node to node v are labeled. If the edge to node $\mathsf{hi}(u)$ or $\mathsf{lo}(u)$ is taken, then f_u is conditioned on (at least) x_u or $\overline{x_u}$, respectively. Note for instance that if $\mathsf{hi}(u) = v$, then $f_{u|x_u} = f_v$. When handling weighted formulas, the weights of the clauses that are satisfied by conditioning f_v on x_v form the set $\mathsf{wg}(v) = \omega_v$ associated with node v. Each root-terminal path contains a variable at most once, and in a particular total or partial order. f_v^p is the arithmetic equivalent to the logical f_v, which in turn, could represent a probability distribution. Extensive definitions and examples of WPBDDs that model probability distributions can be found in [14].

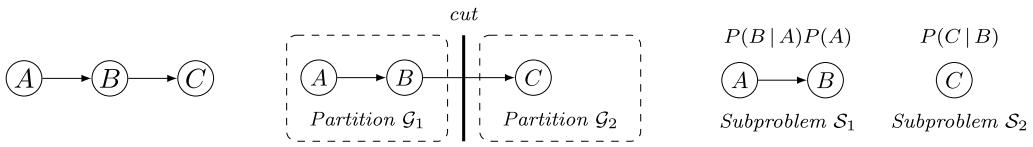
2.5. Removing unintended models

A WPBDD is logically equivalent to the encoding it represents. However, unintended models are present in the encoding (see the remark after Example 2), and thus also in the WPBDD. They are removed during the conversion to the corresponding arithmetic circuit. Luckily, this is achieved with a trivial one-to-one mapping. The conjunctions and disjunctions are mapped to multiplications and additions respectively, and literals ω_i are mapped to probabilities $\mathsf{pr}(\omega_i)$.

There are two requirements on WPBDDs that guarantee that this operation removes the unintended models: (i) the WPBDD must satisfy a particular partial ordering, and (ii) the *delete rule*, which removes nodes that have equivalent children. A partial order is imposed between pairs of literals within a clause, where a literal weighted by 1 must precede a literal weighted otherwise. For example, clause $(\overline{a_1} \vee \overline{b_1} \vee \omega_3)$ produces partial ordering $a_1 \prec \omega_3, b_1 \prec \omega_3$, given that $\mathsf{pr}(a_1) = \mathsf{pr}(b_1) = 1$ and $\mathsf{pr}(\omega_3) \neq 1$. The combined partial ordering produced by all clauses has the effect that all weight literals ω_i are located in the WPBDD after the set of literals by which they are implied, e.g., path $a_1, \ldots, b_1, \ldots, \omega_3$ or $b_1, \ldots, a_1, \ldots, \omega_3$ will exist given the imposed partial ordering. Other weight literals that are distinct from ω_3 but originate from the same CPT are guaranteed not to occur on any path from or to ω_3, because they are implied by a different set of literals, and the delete rule removes their occurrence on any path to and from ω_3. We are left with an arithmetic function that has precisely those models that are considered minimal.

3. The compositional framework

The Weighted Model Counting (WMC) approach is a state-of-the-art method to perform probabilistic inference and has shown to be a powerful tool in practice. However, its wide-spread application is limited due to the cost of one of its core components: compilation. We introduce the *Compositional Framework* (CF): a computational model to perform probabilistic inference that relies on partitioning. The representations of the separate partitions are optimized and when combined with



(a) Bayesian network. **(b)** Partitioning. **(c)** Induced subproblems.

Fig. 5. Partitioning a BN for Example 3.

WMC, CF provides the means to deal with the compilation overhead (see Fig. 2). CF can thus be seen independently of WMC, and is therefore also introduced as such.

3.1. Partitioning

For the purpose of dividing a BN into subproblems, we define a partitioning of graph into subgraphs that are connected components. A partitioning can be obtained by splitting a set of nodes into disjoint subsets of nodes that collectively cover the entire set of nodes, each inducing a connected subgraph, ignoring edges that are not part of the induced subgraphs (i.e. the cut set). Section 6 provides insight into the considerations and mechanics involved in determining a good partitioning.

Definition 7 *(Graph partitioning).* A *partitioning* of a graph $\mathcal{G} = (V, E)$, with nodes V and edges E, is a set of *connected* graphs $\mathcal{G}_1 = (V_1, E_1), \ldots, \mathcal{G}_m = (V_m, E_m)$ induced by a partition $\{V_1, \ldots, V_m\}$ of the vertex set V, i.e., $G_1 = G_{V_1}$, etc. The *cut set* associated with the partitioning is the set of edges connecting subgraphs in the original graph: $E \setminus \bigcup_{i=1}^{m} E_i$.

By combining subgraphs (partitions), their associated variables and probability distributions, we arrive at the complete description of the units of computation, called subproblems.

Definition 8 *(Bayesian network partitioning).* Let $\mathcal{B} = (\mathcal{G}, P)$ be a BN. We say that $\mathcal{B}$ is partitioned into m *subproblems* $S = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ given a partitioning $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$ of graph $\mathcal{G}$, where $\mathcal{G}_i = (V_i, E_i)$. With subproblem $\mathcal{S}_i = (V_i \cup \mathsf{pa}_{\mathcal{G}}(V_i), \mathcal{G}_i, \psi_i)$ is associated a family of conditional probability distributions for each node in V_i and a potential ψ_i defined as follows:

$$\psi_i(X_{V_i}, X_{\mathsf{pa}_{\mathcal{G}}(V_i)}) = \prod_{v \in V_i} P(X_v \mid X_{\mathsf{pa}_{\mathcal{G}}(v)}) \tag{12}$$

The domain of each subproblem S_i is denoted by $\mathsf{dom}(S_i) \triangleq V_i \cup \mathsf{pa}_{\mathcal{G}}(V_i)$.

Example 3 *(Bayesian Network partitioning).* Consider the BN $\mathcal{B} = (\mathcal{G}, P)$ in Fig. 5a. Fig. 5b shows a partitioning $\{\mathcal{G}_1, \mathcal{G}_2\}$, with $\mathcal{G}_1 = (V_1, E_1)$, $V_1 = \{A, B\}$, and $\mathcal{G}_2 = (V_2, E_2)$, with $V_2 = \{C\}$. We use the partitions to define subproblems. Partition $\mathcal{G}_1$ gives rise to subproblem $\mathcal{S}_1 = (Z_1, \mathcal{G}_1, \psi_1)$, and $\mathcal{G}_2$ induces subproblem $\mathcal{S}_2 = (Z_2, \mathcal{G}_2, \psi_2)$ in Fig. 5c, where $Z_1 = V_1 = \{A, B\}$ and $Z_2 = \{B, C\} \neq V_2$. Note that Z_2 also includes B as extra vertex, as the family of conditional probability distributions of C is conditioned on B, or as a potential $\psi_2(B, C)$ (for simplicity's sake we make no distinction between nodes, e.g., A, and variables, e.g., X_A, in the examples).

Lemma 1 *(Partitioned joint distribution). Let BN $\mathcal{B} = (\mathcal{G}, P)$ be partitioned into m subproblems $S = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$, $\mathcal{S}_i = (Z_i, \mathcal{G}_i, \psi_i)$, $\mathcal{G}_i = (V_i, E_i)$. Then it holds that joint probability distribution $P(X_V) = \prod_{i=1}^{m} \psi_i(X_{Z_i})$.*

Proof. The following deduction follows from Definition 8:

$$\begin{aligned} P(X) &= \prod_{v \in V} P(X_v \mid X_{\mathsf{pa}_{\mathcal{G}}(v)}) && \text{(Definition 3)} \\ &= \prod_{i=1}^{m} \prod_{v \in V_i} P(X_v \mid X_{\mathsf{pa}_{\mathcal{G}}(v)}) && (V_1, \ldots, V_m \text{ partitions } V, \text{Definition 8}) \\ &= \prod_{i=1}^{m} \psi_i(X_{Z_i}) && (Z_i = V_i \cup \mathsf{pa}_{\mathcal{G}}(V_i), \text{Definition 8}) \quad \square \end{aligned}$$

Note that each conditional distribution $P(X_v \mid X_{\mathsf{pa}_{\mathcal{G}}(v)})$ in BN $\mathcal{B}$ is associated with one and only one subproblem. A subproblem therefore also depends on the variables associated with the parents in V that are not in the partition $\mathcal{G} = (V, E)$ by which it is induced, i.e., the parents that have been disconnected from their child nodes due to the partitioning operation. Those parents are $X_{\mathsf{pa}_{\mathcal{G}}(V_i)}$.

3.2. Composition

In order to explain how probabilistic inference works using a partitioned probability distribution, a comparison is drawn with the junction-tree algorithm. An important property of junction trees, the graphical representation of the junction-tree message passing algorithm, is the junction property [10,30]. Every node i in a junction tree represents a complete graph (i.e., an undirected graph with all nodes connected to each other) built from a subset of nodes C_i from the original Bayesian network; the *junction property* tells for any pair of junction tree nodes i, j it holds that $C_i \cap C_j \subseteq C_k$ for every node k on the path between i and j [30]. This property ensures that the probability distribution is consistent along every path of the junction tree.

A *composition-tree* ensures probabilistic consistency for CWM in a similar way to how a junction-tree provides these guarantees for the junction-tree algorithm. A composition-tree is induced by a partial ordering based on BN dependencies. It forms a tree-structured graphical representation of how subproblems must be connected, or *composed*, in order to perform inference.

Definition 9 *(Proper composition ordering).* Let BN $\mathcal{B} = (\mathcal{G}, P)$ be partitioned into subproblems S. A partial order $(S, \prec)$ is said to be *proper* if $\mathsf{dom}(S_i) \cap \mathsf{dom}(S_j) \neq \emptyset$ then $\mathcal{S}_i \prec \mathcal{S}_j$ or $\mathcal{S}_j \prec \mathcal{S}_i$ holds, for $i \neq j$.

Definition 10 *(Composition-tree).* Let $(S, \prec)$ be a proper partial ordering for subproblems $S = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ (Definition 9). A *composition-tree* $\mathcal{T} = (S, \mathcal{H})$ is a pair consisting of subproblems S and a directed rooted tree $\mathcal{H} = (U, F)$, with nodes U and edges $F \subseteq U \times U$, such that there is a one-to-one correspondence between nodes U and subproblems S, and there is a directed path from u to v iff $\mathcal{S}_u \prec \mathcal{S}_v$, for $u, v \in U$.

As with junction trees, it is necessary to have consistent probability distributions along the paths of a composition-tree. A composition-tree guarantees that there is a directed path between u and v if the associated subproblems $\mathcal{S}_u$ and $\mathcal{S}_v$ depend on an overlapping set of BN variables. However, unlike with junction-trees, it is not guaranteed that subproblems on a path depend on the variables that the end-point subproblems have in common. The composition-tree is decorated with the notion of *context* (Definition 11). The idea behind the context of a subproblem essentially is to extend the set of variables upon which it depends with precisely those variables that are required for the junction property to hold. In contrast to having this hold for only the nodes on the path between node u to v, this property also holds for endpoint v. This will prove essential during inference, as described in the following section.

Definition 11 *(Composition-tree context).* Let $\mathcal{T} = (S, \mathcal{H})$ be a composition-tree, with tree $\mathcal{H} = (U, F)$, subproblems S, and nodes $Z_u = \mathsf{dom}(S_u)$ of subproblem $S_u \in S$. The *context* of node $v \in U$ is defined as:

$$\mathsf{co}(v) \triangleq \left(X_{Z_v} \cup \bigcup_{u \in \mathsf{ch}(v)} \mathsf{co}(u) \right) \cap \bigcup_{s \in \mathsf{an}(v)} X_{Z_s}. \tag{13}$$

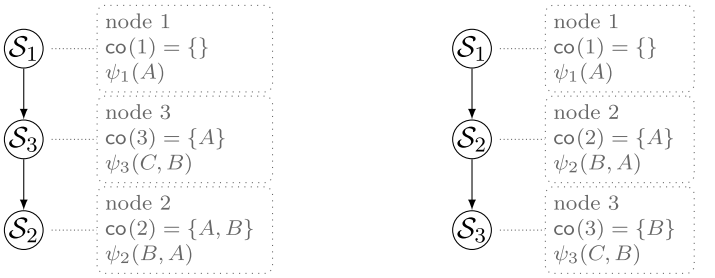
Algorithm 1 creates a (tree structured) composition-tree from a particular sequence of subproblems $\mathcal{S}$ based on the dependencies of the underlying BN. Note that the definition of context in Algorithm 1 differs from Equation (13) in the last term, by making use of the fact that $\mathsf{an}(v) \subseteq \{1, \ldots, m\} \setminus \mathsf{su}(v)$. This change is necessary because the composition-tree is built bottom-up and node v does not have ancestors (yet). The end result is equivalent.

Example 4. Consider the BN $B = (\mathcal{G}, P)$, $\mathcal{G} = (V, E)$, defined over variables $X_V = \{A, B, C\}$ as illustrated in Fig. 5a. The BN is partitioned into 3 subproblems $S = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3\}$, $\mathcal{S}_i = (Z_i, \mathcal{G}_i, \psi_i)$, such that $X_{Z_1} = \{A\}$, $X_{Z_2} = \{A, B\}$, and $X_{Z_3} = \{B, C\}$.

Let $\sigma_1 : \{(1,1), (2,3), (3,2)\}$, $\sigma_2 : \{(1,1), (2,2), (3,3)\}$ and $\sigma_3 : \{(1,2), (2,3), (3,1)\}$ be permutation functions that induce sequences of subproblems S, e.g., $\mathcal{S}_{\sigma_3(1)}, \mathcal{S}_{\sigma_3(2)}, \mathcal{S}_{\sigma_3(3)} = \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1$. Algorithm 1 is used to create composition-trees given these sequences. The following composition-trees (Definition 10) are obtained: $T = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$. $\mathcal{T}_i$ is created by calling CreateCompositionTree$(S, \sigma_i, \mathbf{0})$. Sequence $\mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_1$ for instance induces tree structured composition-tree $\mathcal{T}_3$, which corresponds to partial ordering $\mathcal{S}_2 \prec \mathcal{S}_1, \mathcal{S}_2 \prec \mathcal{S}_3$. $\mathcal{T}_i$ is guaranteed to be a chain (or total ordering) if the final argument to the algorithm is $\mathbf{1}$. Fig. 6 depicts the resulting subproblem orders and shows shared- and context variables.

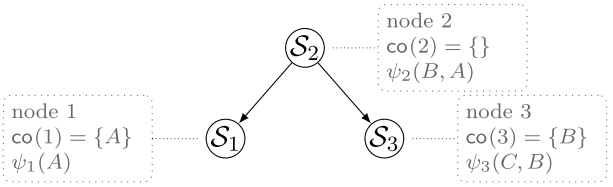
Proposition 1 *(Junction property). Let $\mathcal{T} = (S, \mathcal{H})$ be a composition-tree with graph $\mathcal{H} = (U, F)$ for subproblems S. Function* co *associates precisely those random variables with nodes in U such that $\mathcal{T}$ satisfies the junction property: for each node $s \in U$ on the path from node u to v it holds that $X_{Z_u} \cap X_{Z_v} \subseteq \mathsf{co}(s)$, for $u, v \in U$ and nodes $Z_i = \mathsf{dom}(S_i)$ of subproblem $\mathcal{S}_i \in S$.*

Proof. It follows directly from Definition 10 that there is a path between u and v in tree $\mathcal{H}$ iff the associated subproblems share variables. Each node $v \in U$ has at most one parent, because $\mathcal{H}$ is a rooted tree. Let $u \in \mathsf{an}(v)$, $u, v \in U$. Thus, the sequence $u, s_1, \ldots, s_n, v$ contains all nodes on the path from u to v, with $\{s_1, \ldots, s_n\} = \mathsf{an}(v) \setminus (\mathsf{an}(u) \cup \{u\})$. We must prove that if $X \in X_{Z_u} \cap X_{Z_v}$, then $X \in \mathsf{co}(s_i)$, for all $1 \leq i \leq n$.



(a) Composition-tree $\mathcal{T}_1$ for sequence $\mathcal{S}_1$,$\mathcal{S}_3$,$\mathcal{S}_2$.

(b) Composition-tree $\mathcal{T}_2$ for sequence $\mathcal{S}_1$,$\mathcal{S}_2$,$\mathcal{S}_3$.



(c) Composition-tree $\mathcal{T}_3$ for sequence $\mathcal{S}_2$,$\mathcal{S}_3$,$\mathcal{S}_1$.

Fig. 6. Composition-trees created with Algorithm 1 for Example 4.

Algorithm 1 Create a composition-tree.

CreateCompositionTree(S, σ, b)

input: Subproblems $S = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$, $\mathcal{S}_i = (Z_i, \mathcal{G}_i, \psi_i)$, $\mathcal{G}_i = (V_i, E_i)$, and permutation function σ and Boolean b.
output: Composition-tree $\mathcal{T} = (S, \mathcal{G})$, $\mathcal{G} = (V, E)$.

```
1  def co(v) {
2      return ( X_{Z_v} ∪ ⋃_{u∈ch(v)} co(u) ) ∩ ⋃_{s∈{1,...,m}\su(v)} X_{Z_s}
3  }
4
5  def ord(σ, i) {
6      if (i > |σ|)
7          return ({}, {}, {})
8      else
9          (R, V, E) = ord(σ, i + 1)
10
11         v = σ(i)
12         V' = V ∪ v
13         U = {u ∈ R | X_{Z_v} ∩ co(u) ≠ ∅ ∨ b}
14         E' = E ∪ {(v, u) | u ∈ U}
15         R' = (R\U) ∪ {v}
16         return (R', V', E')
17 }
18
19 (_, V, E) = ord(σ, 1)
20 return (S, (V, E))
```

Firstly, assume that v is a leaf and $X \in X_{Z_u} \cap X_{Z_v}$. Then, $\mathsf{co}(v) = X_{Z_v} \cap \bigcup_{w \in \mathsf{an}(v)} X_{Z_w}$, because $\mathsf{ch}(v) = \{\}$ (Definition 11). It follows that $X \in \mathsf{co}(v)$, because $u \in \mathsf{an}(v)$, $X_{Z_u} \subseteq \bigcup_{w \in \mathsf{an}(v)} X_{Z_w}$ and $X \in X_{Z_u} \cap X_{Z_v}$. This also holds true if v is not a leaf, because the context can only be more inclusive if $\mathsf{ch}(v) \neq \{\}$ (Definition 11). Secondly, $\mathsf{co}(s_n)$ contains at least $\bigcup_{w \in \mathsf{ch}(s_n)} \mathsf{co}(w) \cap \bigcup_{w \in \mathsf{an}(s_n)} X_{Z_w}$ (Definition 11). It follows that $X \in \mathsf{co}(s_n)$, because both the left and right operands of the intersection contain X. Left operand: $v \in \mathsf{ch}(s_n)$, $\mathsf{co}(v) \subseteq \bigcup_{w \in \mathsf{ch}(s_n)} \mathsf{co}(w)$, and $X \in \mathsf{co}(v)$. Right operand: $u \in \mathsf{an}(s_n)$, $X_{Z_u} \subseteq \bigcup_{w \in \mathsf{an}(s_n)} X_{Z_w}$ and $X \in X_{Z_u}$. Finally, for $i = n - 1$ down to 1, it follows that $X \in \mathsf{co}(s_i)$, because $\mathsf{co}(s_i)$ contains $\mathsf{co}(s_{i+1}) \cap \bigcup_{w \in \mathsf{an}(s_n)} X_{Z_w}$, where both operands contain X. $X \in \mathsf{co}(s_n)$, therefore $X \in \mathsf{co}(s_{n-1})$, …, therefore $X \in \mathsf{co}(s_{i+1})$.

Also $X \in \bigcup_{w \in \mathsf{an}(s_n)} X_{Z_w}$ because $u \in \mathsf{an}(s_i)$ and $X \in X_{Z_u}$. We can now state that $X \in \mathsf{co}(s)$ if and only if there exists a u and v such that s lies on the path from u to v and $X \in X_{Z_u} \cap X_{Z_v}$, for $s, u, v \in U$. □

3.3. Inference

Next, we consider how information among conditional probability distributions is shared through message passing in order to perform probabilistic inference in a way that is consistent with the joint probability distribution. Probabilistic inference can generally be performed by variable elimination. The algebraic elimination required to perform inference corresponds exactly with graphical elimination using a composition-tree. Each intermediate step is related to an elimination clique. If we consider a tree of these cliques, then each elimination step can also be thought of as message passing on a clique tree. When CF is combined with WMC, a message is computed by performing weighted model counting on the compiled representation of a particular subproblem. This is discussed in Section 4.

Definition 12 *(Message Passing).* Let BN $\mathcal{B}$ consist of nodes V. BN $\mathcal{B}$ is partitioned into subproblems S that form a composition-tree $\mathcal{T} = (S, \mathcal{H})$ with graph $\mathcal{H} = (U, F)$. We define a sum-product equation for each $u \in U$, which are used as *messages* from u to its parents. A message for each $u \in U$ is defined as:

$$m_u(C) \triangleq \sum_{X_{Z_u} \setminus C} \psi_u(X_{Z_u}) \prod_{k \in \mathsf{ch}_{\mathcal{H}}(u)} m_k(\mathsf{co}(k) \cup C), \tag{14}$$

where $S_u = (Z_u, G_u, \psi_u)$ and $C \subseteq X_V$ are configurations of variables.

Proposition 2 *(Correctness message passing). Let BN $\mathcal{B}$ contain nodes V. BN $\mathcal{B}$ is partitioned into subproblems S that form a composition-tree $\mathcal{T} = (S, \mathcal{H})$ with graph $\mathcal{H}$. If node u is the root of $\mathcal{H}$, then it holds that:*

$$P(E = e) = m_u(e),$$

where $E \subseteq X_V$.

Proof. Claim: Let subproblem $\mathcal{S}_i \in S$ consist of nodes Z_i, i.e., $Z_i = \mathsf{dom}(S_i)$. Suppose we take some node i in $\mathcal{H}$, where i and its descendants in $\mathcal{H}$ represent conditional distributions Ψ_i over a set of variables $X_{\Psi_i} \subseteq X_V$, then for any $E \subseteq X_{\Psi_i}$ it holds that m_i is a potential function over E such that:

$$m_i = \sum_{X_{\Psi_i} \setminus E} \prod_{\psi \in \Psi_i} \psi$$

This can be proven by structural induction on the subtree of $\mathcal{H}$ induced by i and its descendants. Suppose first that i is a leaf node in $\mathcal{H}$, then the property follows directly from Definition 12. Now suppose that i is not a leaf node, then by the induction hypothesis, we have:

$$m_i = \sum_{X_{Z_i} \setminus E} \psi_i \prod_{k \in \mathsf{ch}_{\mathcal{H}}(i)} \sum_{X_{\Psi_k} \setminus (\mathsf{co}(k) \cup E)} \prod_{\psi \in \Psi_k} \psi$$

Observe that $(X_{\Psi_k} \setminus (\mathsf{co}(k) \cup E)) \cap (X_{Z_i} \setminus E) = (X_{\Psi_k} \setminus \mathsf{co}(k)) \cap X_{Z_i} = \varnothing$ because variables from X_{Z_i} are part of $\mathsf{co}(k)$ if they occur in X_{Ψ_k} (Proposition 1). Furthermore, because the tree is induced by a *proper* ordering, each child will marginalize over distinct sets of variables. Observe that $\bigcup_k (X_{\Psi_k} \setminus \mathsf{co}(k)) \cup X_{Z_i} = X_{\Psi_i}$. It follows that:

$$m_i = \sum_{X_{Z_i} \setminus E} \sum_{\substack{X_{\Psi_k} \setminus \mathsf{co}(k), \\ k \in \mathsf{ch}_{\mathcal{H}}(i)}} \psi_i \prod_{k \in \mathsf{ch}_{\mathcal{H}}(i)} \prod_{\psi \in \Psi_k} \psi = \sum_{X_{\Psi_i} \setminus E} \prod_{\psi \in \Psi_i} \psi$$

Finally note that for root node u it holds by Lemma 1 that $P(X_V) = (\prod_{\psi \in \Psi_u} \psi)(X_V)$. Hence, $P(e) = (\sum_{X_\Psi \setminus E} \prod_{\psi \in \Psi_u} \psi)(e) = m_u(e)$. □

Example 5 *(Message passing).* Consider the partitioning in Example 4. Fig. 7 illustrates message passing on composition-tree $\mathcal{T}_3 = (S, \mathcal{H})$ (Fig. 6c) based on Definition 12. Example, for 6c, without evidence the messages are if $e = \varnothing$:

$$m_1(A) = P(A)$$
$$m_3(B) = \textstyle\sum_C P(C \mid B)$$
$$m_2(\varnothing) = \textstyle\sum_{A,B} P(B \mid A) m_1(A) m_3(B)$$

so we find that $P(\Omega) = m_2(\varnothing) = 1$.

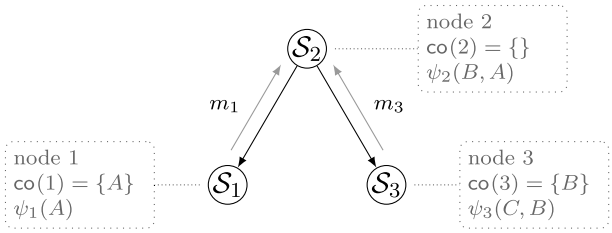


Fig. 7. Message passing for Example 5.

Now suppose $e = \{A = a\}$, then

$$m_1(a) = P(a)$$
$$m_3(B) = \sum_C P(C \mid B)$$
$$m_2(a) = \sum_B P(B \mid a) m_1(a) m_3(B)$$

It follows that: $m_2(a) = \sum_B P(B \mid a) m_1(a) m_3(B) = \sum_B P(B \mid a) P(a) \sum_C P(C \mid B) = P(a)$.

The key insight here is that messages among subproblems can be reused, allowing you to pre-compute clique functions/factors, and get new marginals quickly, as conditional probabilities are represented as clique potentials. A natural ordering is to eliminate from the bottom leaves to the top. However, a composition-tree can be re-ordered with a different node as root. Each message is the factor resulting from eliminating variables in descendant subproblems.

4. Compositional weighted model counting

Compositional inference (CI) can be instantiated with any method that computes marginal probabilities. Combined, the subproblems that make up a composition-tree represent a probability distribution, where each subproblem represents a disjoint portion of it. Inference consists of so-called edge computations, performed by message passing (Definition 12), and node computations, which can be outsourced to a method that computes marginals. Weighted Model Counting (WMC) is one such method.

The Achilles' heel of WMC has always been the cost of knowledge compilation, which is exponential complexity in the worst case [3]. WMC therefore is a much less favored method for problems of larger size, despite its acclaimed high inference efficiency after compilation. To this end, we introduce *Compositional Weighted Model Counting* (CWMC), which combines CF and WMC. CWMC allows for a divide and conquer strategy, where compiling multiple partitions can potentially be much less costly compared to monolithic (or unpartitioned) compilation.

Partitioning has several advantages: (1) reduced compilation cost, and (2) improved capability of exploiting local structure through independent compilation orderings for each subproblem. To elaborate, Example 3 provides a case where, in the worst case, compilation is exponential in $\{A, B, C\}$. Through partitioning this reduces to exponential in $\{A, B\}$ and $\{B, C\}$. Now consider the extent to which compilation cost is reduced for problems of a much larger size. Partitioning thus provides a scalable way to tackle compilation scenarios of high cost.

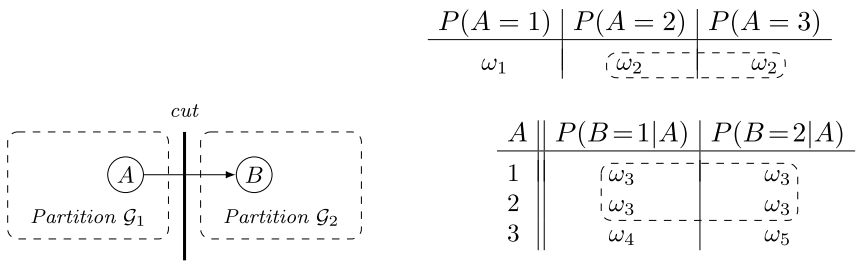
Secondly, the degree to which the problem structure can be exploited is determined by the ordering used during compilation. An ordering that is good for one part of the network, might not be well suited for another. We improve upon this by allowing subproblems to be compiled with independent orderings, rather than having one global ordering. This allows for more fine-grained control to exploit structure and capturing network topology.

4.1. Inference by CWMC

Probabilistic inference is achieved by 4 distinct phases (also see Fig. 2).

1. *Partitioning*: Partition a BN into m subproblems.
2. *Compilation*: Encode and compile each subproblem.
3. *Composition*: Compose compiled subproblems.
4. *Inference*: Perform inference by WMC using the composed representation.

We provide a step by step walkthrough, where each phase is explained and demonstrated with examples. Insight is provided into the advantages of partitioning in relation to WMC. The required additional considerations to retain a consistent model count with regard to the probability distribution are introduced. The presented framework reduces compilation cost, which allows WMC methods to be applied to practically any BN. WMC representations are supported that are less or equally succinct as decision-DNNF. This includes for instance SDDs, OBDDs, ZBDDs, WPBDDs, and others. This is because these representations can be converted to an equivalent Free Binary Decision Diagram (FBDD) [2] (Section 7). Without loss of generality, a particular combination of encoding and target representation is used for demonstration purposes, producing one variety of decision diagram, called *Weighted Positive Binary Decision Diagram* (WPBDD) [14].



(a) Partitioning. **(b)** Local structure in CPTs.

Fig. 8. Partitioning a BN for Example 6 and 7.

4.2. Partitioning and compilation

Any BN can be partitioned using the principles introduced in Section 3.1. Once a BN is partitioned into subproblems, a Boolean encoding must be obtained. This conversion allows the use of state-of-the-art compilation methods, which in turn allows for efficient probabilistic inference by weighted model counting. The used Boolean encoding is presented in Section 2.3.2.

Example 6 *(Encoding a partitioned BN).* Consider the BN $\mathcal{B} = (\mathcal{G}, P)$ from Example 1. Fig. 8a shows a partitioning $\{\mathcal{G}_1 = (V_1, E_1), \mathcal{G}_2 = (V_2, E_2)\}$ of $\mathcal{G}$ that respectively induces subproblems $S = \{\mathcal{S}_1, \mathcal{S}_2\}$, where $V_1 = \{A\}$ and $V_2 = \{B\}$. $\mathcal{S}_1$ thus depends on $X_{\mathsf{fa}_{\mathcal{G}}(V_1)} = \{A\}$, and $\mathcal{S}_2$ depends $X_{\mathsf{fa}_{\mathcal{G}}(V_2)} = \{A, B\}$, respectively.

The probabilities of CPTs associated with each subproblem are encoded such that equal probabilities in a particular CPT are mapped to the same Boolean atom. This is shown in Fig. 8b. A standalone encoding for subproblem $\mathcal{S}_1$ and $\mathcal{S}_2$ is given below by Boolean functions f and g, respectively. Note that constraints for variable A are mentioned in both f and g (see line 1 of both formulas), because both subproblems depend on A

$$
\begin{aligned}
f \;=\; & (a_1 \vee a_2 \vee a_3) \;\wedge\; (\overline{a_1} \vee \overline{a_2}) \;\wedge\; (\overline{a_1} \vee \overline{a_3}) \;\wedge\; (\overline{a_2} \vee \overline{a_3}) \wedge && 1\\
& (\overline{a_1} \vee \omega_1) \;\wedge\; (\overline{a_2} \vee \omega_2) \;\wedge\; (\overline{a_3} \vee \omega_2). && 2\\
g \;=\; & (a_1 \vee a_2 \vee a_3) \;\wedge\; (\overline{a_1} \vee \overline{a_2}) \;\wedge\; (\overline{a_1} \vee \overline{a_3}) \;\wedge\; (\overline{a_2} \vee \overline{a_3}) \wedge && 1\\
& (b_1 \vee b_2) \;\wedge\; (\overline{b_1} \vee \overline{b_2}) \wedge && 2\\
& (\overline{a_1} \vee \overline{b_1} \vee \omega_3) \;\wedge\; (\overline{a_1} \vee \overline{b_2} \vee \omega_3) \;\wedge\; (\overline{a_2} \vee \overline{b_1} \vee \omega_3) \wedge && 3\\
& (\overline{a_2} \vee \overline{b_2} \vee \omega_3) \;\wedge\; (\overline{a_3} \vee \overline{b_1} \vee \omega_4) \;\wedge\; (\overline{a_3} \vee \overline{b_2} \vee \omega_5). && 4
\end{aligned}
$$

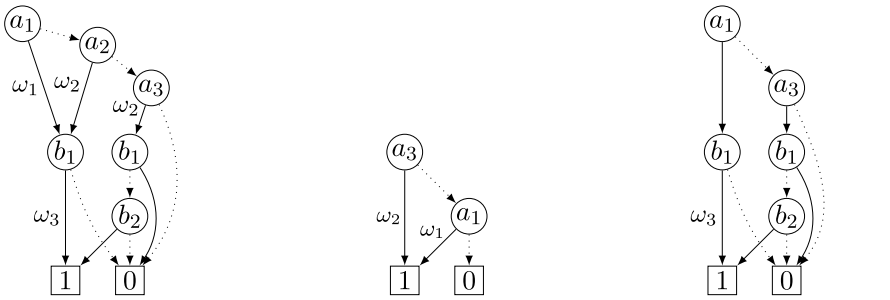
Typically, and also in the case of WPBDDs, compilation is driven by an ordering on the variables. With CWMC, each subproblem is compiled individually, thus each subproblem requires an ordering on the variables upon which it depends. CWMC allows the use of independent- and even conflicting compilation orderings among subproblems.

Example 7 *(Compiled representations of a partitioned BN).* Consider the subproblems and their encoding formulas f and g from Example 6. Fig. 9 shows compiled representation induced by conflicting orderings. Note that if an edge is weighted by an atom that maps to probability one, then the weight is removed. If a weight maps to probability zero, then the weight is removed and the edge's target is changed to the **0**-terminal. Hence, weights ω_4 and ω_5 are not present for this reason. Details of the compilation process can be found in [14], or in the respective articles that introduce the otherwise preferred symbolic target representation.

4.3. Composition

Probabilistic inference by WMC is traditionally performed by traversing the representation obtained through compilation, whilst computing the underlying arithmetic function it actually represents. This achieves marginalization. The arithmetic function is unique to the symbolic target representation that is chosen and is best explained by the articles that introduce them.

With CWMC, we must somehow traverse the collection of compiled subproblems as if it were a monolithic representation. We do this by composing the compiled subproblems according to the structure of a composition-tree. Every child-parent pair is composed by connecting the **1**-terminal of the parent subproblem to the root node of the child subproblem. Thus allowing a traversal across subproblems.

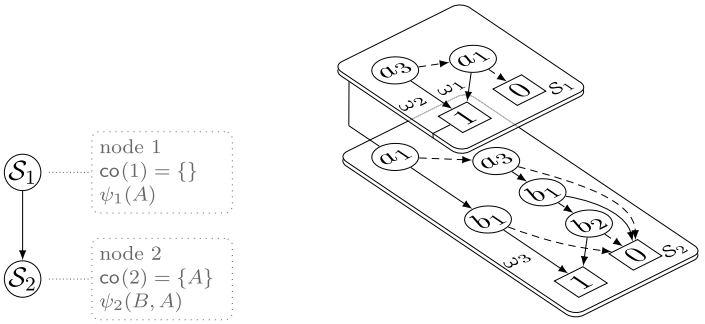


(a) Unpartitioned BN $\mathcal{B}$, compiled with order: $a_1 \prec a_2 \prec a_3 \prec b_1 \prec b_2$.

(b) Subproblem $\mathcal{S}_1$, compiled with order: $a_3 \prec a_2 \prec a_1$.

(c) Subproblem $\mathcal{S}_2$, compiled with order: $a_1 \prec a_2 \prec a_3 \prec b_1 \prec b_2$.

Fig. 9. Compiled representations of a partitioned BN for Example 7, where x_i signifies x being equal to its i^{th} value.



(a) Composition-tree $\mathcal{T}$.

(b) Composed representation.

Fig. 10. Composed representation for Example 8 and 11.

Example 8 *(Subproblem composition).* Consider the compiled subproblems of Example 7. Fig. 10 shows composition-tree $\mathcal{T}$ induced by sequence $\mathcal{S}_1$,$\mathcal{S}_2$ (Fig. 10a), and the composed representation that has a one-to-one mapping with $\mathcal{T}$ (Fig. 10b).

4.4. Inference

The typical WMC process compiles a BN monolithically. The obtained compiled representation has an underlying arithmetic circuit that represents the sum-product formula in a factored form. This factored form can be evaluated with improved efficiency if it is more concise than the original. The evaluation is performed while traversing the compiled representation.

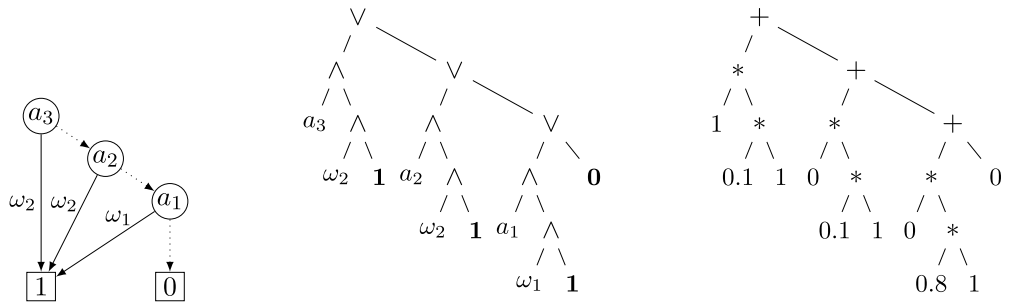
4.4.1. Traversing monolithic representations

If a symbolic language is chosen that is decomposable, deterministic and smooth, then any path from root to leaf contains a particular variable at most once [18]. A path in a compiled representation encodes a configuration, or assignment to all contained variables. Taking the hi or lo edge at node v implies the assignment $x_v = \mathbf{1}$ or $x_v = \mathbf{0}$, respectively (see Fig. 4b). If $x_v = \mathbf{1}$, then evidence $\mathsf{bnvar}(x_v) = \mathsf{bnval}(x_v)$ is semantically implied (see Definition 5). This way, we can sum over all configurations of a BN and obtain the joint distribution.

Example 9 *(Traversing a compiled subproblem).* Consider only variable A in Example 1. The compiled representation is shown in Fig. 11a. We have not optimized the representation in order to make the upcoming discussion easier. Otherwise, it would look like Fig. 9b. The underlying logical circuit is shown in Fig. 11b (obtained with the circuit in Fig. 4c), and an instantiated arithmetic circuit given evidence $A = 3$ is shown in Fig. 11c.

Each path from the root to the **1**-terminal semantically implies evidence. There are three possible paths shown below. If we have evidence prior to traversing the compiled representation, we only consider the paths that are consistent with the evidence.

Path	Logic	Meaning
$a_3 \rightarrow \mathbf{1}$	$\overline{a_1} \wedge \overline{a_2} \wedge a_3$	$A = 3$
$a_3 \dashrightarrow a_2 \rightarrow \mathbf{1}$	$\overline{a_1} \wedge a_2 \wedge \overline{a_3}$	$A = 2$
$a_3 \dashrightarrow a_2 \dashrightarrow a_1 \rightarrow \mathbf{1}$	$a_1 \wedge \overline{a_2} \wedge \overline{a_3}$	$A = 1$



(a) Subproblem $\mathcal{S}_1$. (b) Logical circuit. (c) Arithmetic circuit.

Fig. 11. Performing inference in Example 9.

4.4.2. Traversing partitioned representations

Here, a BN is not compiled monolithically but partitioned. WMC entails traversing several compiled representations, given the structure of a composition-tree. A path that ends in the **1**-terminal of a parent subproblem can continue at the root node of a child subproblem. The formula that is evaluated is provided by Definition 12. Although a directed path in a compiled representation can contain a particular variable only once, it can still happen that a variable occurs multiple times on a path that stretches across multiple subproblems. This could lead to an inconsistent assignment (multiple values are assigned to the same variable), which leads to an inconsistent model count.

Given a composition-tree, if a parent subproblem is traversed and the path leading to the **1**-terminal semantically implies evidence $A = 1$, then the child subproblem must be traversed in such a way that all paths leading to its **1**-terminal are consistent with $A = 1$. No path must be allowed to semantically imply $A \neq 1$. Continuing the traversal of the parent subproblem will lead to paths that semantically imply $A = 2$, $A = 3$, and so on. Each time, the child subproblem must be traversed in a way that is consistent with its parent's evidence. If variable A has three possible values, then the child subproblem must be traversed three times.

More generally, consistency is obtained by *dynamic conditioning*: given a composition-tree, we remove the models in the child subproblem that semantically imply evidence that is inconsistent with *ancestor evidence*: the evidence that is semantically implied by the path that stretches across all ancestor subproblems. This can be achieved by simply pruning the paths in a compiled representation that lead to an inconsistent model count, or explicitly conditioning the compiled representation on ancestor evidence. An algorithm for dynamic conditioning and an example is introduced later.

4.4.3. Consistent model counts

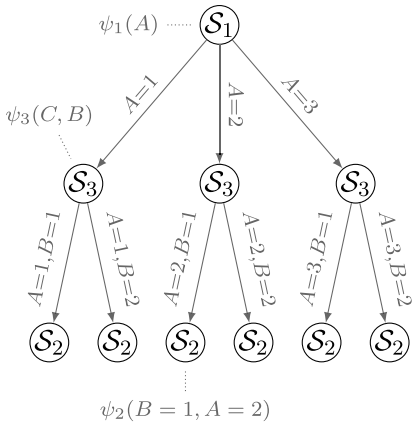
A key insight is that dynamic conditioning involves precisely those variables that are in a subproblem's context. Context is provided by a composition-tree. Also, any path that leads to a child subproblem, across its ancestor subproblems, instantiates the child's context variables. Thus prior to the child's traversal, all context variables are known. The number of messages that are sent from a child to a parent subproblem is equal to the number of configurations of the child's context variables.

A *message graph* is introduced to capture this message passing structure, and expose opportunities for previously computed messages to be reused.

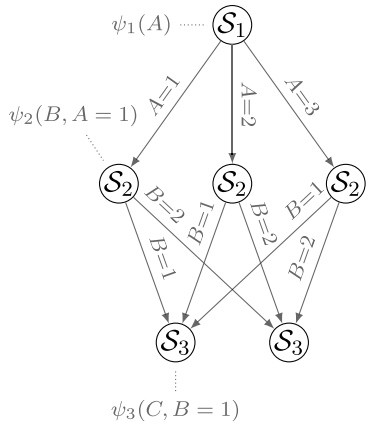
Definition 13 *(Message graph).* Let $\mathcal{T} = (S, \mathcal{H})$, $\mathcal{H} = (U, F)$ be a composition-tree for the set of subproblems S. A *message graph* $\mathcal{U} = (S, \mathcal{I})$ is a rooted DAG $\mathcal{I} = (W, H)$ such that there is a one-to-many mapping from nodes U to nodes W, with edges $H \subseteq W \times W$. Message graph $\mathcal{U}$ is induced by $\mathcal{T}$ such that:

1. Node $u \in U$ maps to nodes Y, where $Y \subseteq W$. For each node $y \in Y$ we define:
   (a) Node y is labeled by $\mathcal{S}_u$;
   (b) Context $\mathsf{co}(y) \triangleq \mathsf{co}(u)$;
   (c) Evidence e_y is associated with y, which is a distinct configuration of context $\mathsf{co}(y)$, i.e., $e_y \neq e_z$ with $y \neq z$ and $z \in Y$. Note that $|Y| = \mathsf{car}(\mathsf{co}(u))$.
2. Edge $(y, z) \in W$ iff $(u, v) \in F$ and e_y is consistent with e_z, where $u, v \in U$, $u \neq v$, u maps to Y and v maps to Z, $Y \subset W$, $Z \subset W$, $y \in Y$ and $z \in Z$, i.e., e_y has the same configuration as e_z for variables $\mathsf{co}(y) \cap \mathsf{co}(z)$.

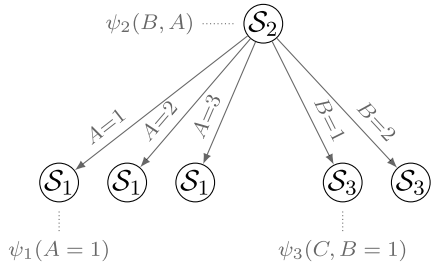
A message graph has a node v for each unique configuration e_v of a subproblem's context $\mathsf{co}(v)$. This configuration is used for dynamic conditioning. Alternatively, a message graph has a node for each summation term computed by Definition 12. Each edge represents the passing of a message, while each node represents the computation of the message based on dependent incoming messages from its descendants. A node's computation can be reused by each of its parents. A node with many parents thus has a distinct advantage over a node with few parents.



(a) Message graph $\mathcal{U}_1$ for the composition-tree $\mathcal{T}_1$ (Fig. 6a).



(b) Message graph $\mathcal{U}_2$ for the composition-tree $\mathcal{T}_2$ (Fig. 6b).



(c) Message graph $\mathcal{U}_3$ for the composition-tree $\mathcal{T}_3$ (Fig. 6c).

Fig. 12. Message graphs for Example 10.

Example 10 *(Message graphs).* Consider composition-trees $T = \{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3\}$ from Example 4. Fig. 12 shows message graphs $U = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3\}$ induced by respective composition-trees T. Each node v in a message graph represents a subproblem $\mathcal{S}_v$ given distinct evidence e_v, with $E = \mathsf{co}(v)$. This evidence is depicted as labels on node v's incoming edges.

Algorithm 2 implements dynamic conditioning, based on the structure of a message graph, in order to compute marginal probability $P(E = e)$ given evidence e. Two structures are traversed to compute this probability: the message graph, and the compiled subproblems associated with message graph nodes. Consider the message passing formula in Definition 12. The two structures are responsible for computing a distinct part:

$$m_u(E) \triangleq \underbrace{\sum_{X_{Z_u} \setminus E} \varphi_u(X_{Z_u})}_{\substack{\text{Traverse} \\ \text{compiled} \\ \text{representation}}} \underbrace{\prod_{k \in \mathsf{ch}_{\mathcal{U}}(u)} m_k(\mathsf{co}(k) \cup E)}_{\substack{\text{Traverse} \\ \text{message graph}}}$$

Algorithm 2 incorporates evidence when computing the probability for a node with function ComputeNodeProbability. For example, let evidence $A = a$, node v is labeled by Boolean x_v, $\mathsf{bnval}(x_v) = a$ and $\mathsf{bnvar}(x_v) = A$. In this instance we can simply multiply the probability computed for the lo edge by 0, and the probability of the hi edge by 1. How ComputeNodeProbability computes a probability depends on the symbolic language that is used. For WPBDDs one can look at Fig. 4c and [14].

Note that *Algorithm 2* is naive in that is does no caching at all. This is trivially added for both the message graph and the compiled representations, by simply storing the nodes that have been visited, and using the probability that was previously computed upon encountering a visited node.

Example 11 *(Traversing the composed representation).* Consider the composed representation from Fig. 10b. Fig. 13a shows the corresponding message graph $\mathcal{U}$. The subproblems are traversed according to the structure of message graph $\mathcal{U}$.

Algorithm 2 Marginal inference, where directed graph $C_i = (U, F)$ is the compiled representation of subproblem S_i, with nodes U and edges $F = U \times U$.

ComputeProbability($\mathcal{U}, e$)

input: Message graph $\mathcal{U} = (S, \mathcal{I})$, with graph $\mathcal{I}$, and evidence Q
output: Marginal probability $P(E = e)$ given evidence e

```
1  r = rt_I
2  return TraverseSubproblem(rt_{C_r}, r, e)
```

TraverseMessageGraph(w, e)

input: Message graph node w and evidence e.
output: Marginal probability given evidence e

```
1  R = {i ∈ ch(w) | e_i is consistent with e}
2  return Σ_{r∈R} TraverseSubproblem(rt_{C_r}, r, e)
```

TraverseSubproblem(v, w, e)

input: Node v of compiled representation C_w, message graph node w and evidence e
output: Marginal probability of subproblem $\mathcal{S}_w$ given evidence e

```
1  if x_v == 0            // false terminal
2      return 0
3  elseif x_v == 1        // true terminal
4      return TraverseMessageGraph(w, e)
5  else                   // internal node
6      p_hi = TraverseSubproblem(hi(v), w, e ∪ bnvar(x_v) = bnval(x_v)) // add evidence
7      p_lo = TraverseSubproblem(lo(v), w, e)
8      p = ComputeNodeProbability(p_hi, p_lo, e) // language dependent arithmetic function
9      return p
```

Assume no evidence. We start traversal at root node v in message graph $\mathcal{U}$. Note that $\mathcal{S}_v = S_1$. Consider the possible paths leading to the **1**-terminal previously mentioned in Example 9, e.g., $a_3 \to \mathbf{1}$ has semantic meaning $A = 3$.

We select the set of children of v that are consistent with the configuration determined by the traversal of the parent. If $\mathcal{S}_v$ is traversed using path $a_3 \to \mathbf{1}$, then we have to select all child subproblems consistent with $A = 3$. Fig. 13 shows all possible children, where edges are crossed out, and nodes are grayed out, that should not be traversed. Thus for path $a_3 \to \mathbf{1}$ only one child is selected (shown in Fig. 13d). The traversal of $\mathcal{S}_1$ and $\mathcal{S}_2$ in this way computes $P_1(A = 3)P_2(B \mid A = 3)$. Continuing the traversal computes $P_1(A)P_2(B \mid A)$.

4.4.4. Combining components

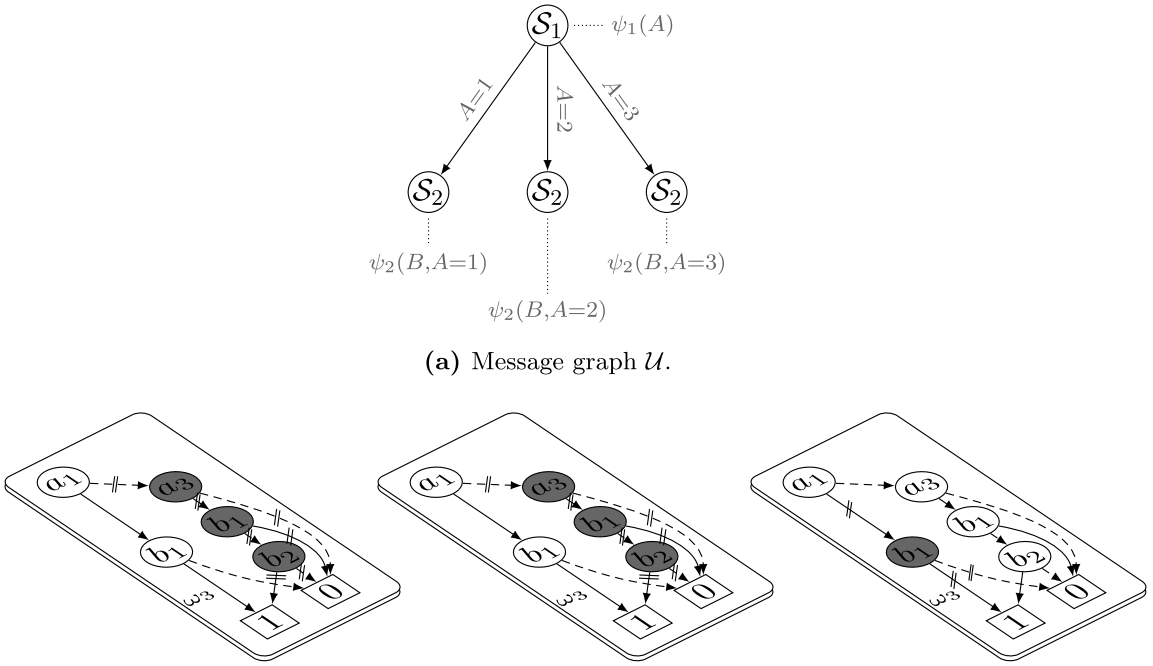
For convenience, we now bring together the core components of CWMC in Definition 14: A BN is partitioned into a set of subproblems; The subproblems are partially ordered based on the structure of the underlying BN (a *proper* partial ordering) that induces a composition-tree; The composition-tree induces a message graph; Inference is performed using message passing on the message graph.

Definition 14 *(Composition problem).* A *composition problem* is a tuple $(\mathcal{B}, S, \mathcal{T}, \mathcal{U})$ defined as follows:

1. $\mathcal{B} = (\mathcal{G}, P)$ is a BN, with $\mathcal{G} = (V, E)$, defined over variables X_V (Definition 3);
2. S is the set of subproblems by partitioning $\mathcal{B}$ (Definition 8);
3. $\mathcal{T}$ is a composition-tree for subproblems S (Definition 10);
4. $\mathcal{U}$ is a message graph induced by $\mathcal{T}$ (Definition 13).

The compositional algorithm. Algorithm 3 implements the compositional approach. It takes as input a Bayesian network $\mathcal{B}$ defined over variables X and joint probability queries Q of the form $P(E = e)$ with $E \subseteq X$). CompositionalInference computes the marginal probability $P(E = e)$ in four steps:

1. **Partitioning**. A BN $\mathcal{B}$ is partitioned into m subproblems $S = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ (Partition and CreateSubproblems);
2. **Compilation**. Each subproblem $\mathcal{S}_i$ is compiled to a representation C_i, given an ordering σ that is determined independently for each subproblem (DetermineCompilationOrdering, Algorithm 4);



(a) Message graph $\mathcal{U}$.

(b) Traversing $\mathcal{S}_2$ when $A = 1$, i.e., $\psi_2(B, A = 1)$ **(c)** Traversing $\mathcal{S}_2$ when $A = 2$, i.e., $\psi_2(B, A = 2)$ **(d)** Traversing $\mathcal{S}_2$ when $A = 3$, i.e., $\psi_2(B, A = 3)$

Fig. 13. Traversing the composed representation for Example 8 and 11.

Algorithm 3 The compositional framework.

CompositionalInference($\mathcal{B}$, Q)

input: Bayesian network $\mathcal{B}$, and queries Q of the form $P(E = e)$, with $E \subseteq X_V$ and V the nodes of $\mathcal{B}$.
output: The marginal probability of each query in Q

```
// Phase 1: partitioning
cut = Partition(B)
S = CreateSubproblems(B, cut)

// Phase 2: compilation
for S_i ∈ S
    σ = DetermineCompilationOrdering(S_i)
    K = CreateCompilationTree(B, S_i, σ, 0) // (Algorithm 4)
    C_i = CompileSubproblem(S_i, K)

// Phase 3: composition
σ_S = DetermineSubproblemOrdering(S)
T = CreateCompositionTree(S, σ_S, 0) // (Algorithm 1)
U = CreateMessageGraph(S, T) // (see Definition 13)
C = (B, S, T, U) // composition problem

// Phase 4: inference
return ⋃_{e∈Q} ComputeProbability(U, e) // (Algorithm 2)
```

3. **Composition**. A composition-tree $\mathcal{T}$ is created (CreateCompositionTree, Algorithm 1) for message graph $\mathcal{U}$ (CreateMessageGraph). Composition problem $\mathcal{C}$ is constructed;
4. **Inference**. Inference is (repeatedly) performed by traversing the compiled subproblems (ComputeProbability, Algorithm 2). They are composed according to $\mathcal{T}$, and traversed according to message graph $\mathcal{U}$.

Table 1
(Tree-)widths of the composition-trees in Fig. 6 for Example 12.

	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$
$w(v_1)$	3	3	5
$w(v_2)$	6	6	0
$w(v_3)$	0	0	0
$w(\mathcal{T}_i)$	6	6	5

5. The cost of compositional inference

Probabilistic inference remains NP-hard, even in the approximate inference case [11]. Despite that, Bayesian network inference algorithms have been highly successful in practical situations, by providing favorable runtime complexities when measures of complexity of a network are bounded. CF and its message passing are inspired by the well-established theories behind the junction-tree algorithm, for which *tree-width* is often used as the measure for complexity.

Definition 15 *(Tree-width).* [19] Given an ordered graph $\mathcal{G} = (V, E)$, the *width* of a node in an ordered graph is the number of neighbors that precede it in the ordering. The *tree-width* of an ordering is the maximum width over all nodes. The *path-width* of a graph is the tree-width over the restricted class of orderings that corresponds structurally to a chain (i.e., total orderings).

It is known that partial orderings (i.e., those that graphically resemble a tree) induce representations that enjoy tighter size upper bounds (based on tree-width) compared to total orderings (based on path-width) [19]. A composition-tree resembles a tree and induces a message graph. There are potentially a very large number of possible message graphs given a particular partitioning. The size of a message graph determines how many messages are passed, thus the tree-width of the inducing composition-tree provides implications on the cost of inference.

The cost of inference is proportional to the tree-width of a particular partitioning. The tree-width of a partitioning is equal to the lowest tree-width among all possible composition-trees, and the tree-width of a composition-tree is given in Definition 16.

Definition 16 *(Composition tree-width).* Let $\mathcal{T} = (S, \mathcal{H})$, $\mathcal{H} = (U, F)$, be a composition-tree for subproblems S. The width of node $v \in U$ is defined as:

$$w(v) = \sum_{u \in \mathsf{pa}(v)} \mathsf{car}(\mathsf{co}(v)) \cdot \mathsf{car}(\mathsf{co}(u)). \tag{15}$$

The *tree-width* of composition-tree $\mathcal{T}$ is defined as:

$$w(\mathcal{T}) = \max\{w(v) \mid v \in U\}. \tag{16}$$

Example 12 *(Tree-width).* Consider the composition-trees $\mathcal{T}_1, \mathcal{T}_2$ and $\mathcal{T}_3$ of Example 4 that induce the message graphs $\mathcal{U}_1, \mathcal{U}_2$ and $\mathcal{U}_3$ in Example 10, respectively. The node- and tree-widths of each of the composition-trees are listed in Table 1. Recall that the number of edges in an induced message graph is equal to the number of messages that are passed during inference. The composition-tree with the lowest tree-width coincides with the message graph with the least number of edges, i.e., ordering $\mathcal{T}_3$ and message graph $\mathcal{U}_3$.

The compilation bottleneck can effectively be tackled by CF, but the cost of inference *might* increase in comparison. Ideally, a partitioning is sought that optimizes both compilation time and inference time. We aim to provide insight into the relation between the two here. Inference cost is influenced by (1) the quality of the partitioning, in turn determined by cutset size, (2) the size of the message graph induced by the composition-tree, and (3) the variable orderings used to compile individual subproblems.

CWMC is especially beneficial to symbolic target representations that are totally ordered. A hybrid form is created where a message graph, that is tree ordered, consists of target representations, that are totally ordered. In this sense, these target representations are extended by relaxing their total order into a partial order if the underlying BN allows it. This reduces representation size as upper bounds on representation size are tighter based on tree-width, compared to path-width [19]. Inference time complexity is reduced to $\mathcal{O}(n \exp(w))$, where $n = |X|$, X a set of BN variables, and w is the tree-width of composition ordering $\mathcal{T}$.

A target representation is said to be deterministic if the disjuncts of any disjunction are pairwise logically inconsistent [18]. OBDDs are deterministic because no decision can lead to ambiguity, thus each variable occurs only once on every

path from root to leaf. Partitioning violates this property and thereby introduces non-determinism. The possible increase of inference cost stems from the relaxation of determinism in the used target representation, imposed by the partitioning. The more variables are shared among subproblems, the more redundant decisions will have to be made (through dynamic conditioning).

Determinism, however, was found to be a necessary condition for the ability to perform inference in linear time [18]: While allowing non-determinism could potentially lead to more concise representations with polynomial time transformations, many other operations, such as inference, become intractable. Our compositional approach relies on a restricted form of *non-deterministic partitioning* [1]. The restriction imposed here is that the smallest possible partitioned probability space leaves conditional distributions intact.

Theorem 1 *(Inference complexity). Let $\mathcal{C} = (\mathcal{B}, S, \mathcal{T}, \mathcal{U})$ be a composition problem. The compositional approach offers inference time complexity $\mathcal{O}(n \exp(w))$, where $n = |X|$ and w is the tree-width of $\mathcal{T}$, if the symbolic target representation chosen for compilation is a subset of d-DNNF.*

Proof. First we show that inference is of linear time complexity in the size of message graph $\mathcal{U}$, if the symbolic target representation chosen for compilation is a superset of d-DNNF, i.e., it contains the properties of decomposability and determinism, which have been identified as key representational axioms for tractable (linear-time) probabilistic inference. Various target representations adhere to these axioms, such as WPBDD, OBDD, and d-DNNF, where d-DNNF is the most general representation that includes these two properties. Consider the propositional interpretation of these properties [18]:

- *Decomposable*: For each conjunction $f^1 \wedge \cdots \wedge f^n$, the conjuncts $f^1, \ldots, f^n$ do not share variables;
- *Determinism*: For each disjunction $f^1 \vee \cdots \vee f^n$, every pair of disjuncts are logically contradictory, i.e., $f^i \wedge f^j = \mathbf{0}$, for $i \neq j$.

Some representations such as WPBDD also have the decision property, which states:

- *Decision*: Each disjunction is of the form $(x \wedge f_x) \vee (\overline{x} \wedge f_{\overline{x}})$, where x is a variable and $f_x, f_{\overline{x}}$ the positive and negative cofactors of f given x.

Note that any representation that satisfies the decision property is also deterministic.

We claim that the message graph satisfies the decision and decomposability axioms. Each internal node represents a decision on an attribute, and each branch represents a distinct outcome of that decision. The decision here relates to the configuration of context variables. Variables that are shared and additionally occur in a subproblem's context are encountered more than once from root to leaf. Decomposibility is therefore ensured by dynamic conditioning. It follows that if the target representation supports linear time model counting, that inference via model counting also requires linear time using the composite representation.

Secondly, we prove that inference is of time complexity $\mathcal{O}(n \exp(w))$. Assume we have partitioned BN $\mathcal{B} = (\mathcal{G}, P)$, $\mathcal{G} = (V, E)$, into $|V|$ subproblems. Then, there is a one-to-one correspondence between the nodes of the composition-tree $\mathcal{T} = (U, F)$ and the BN. Each node $u \in U$ then is a subproblem that represents a distribution coinciding with a CPT in the BN (Definition 8). The number of times the CPT associated with u must be (dynamically) conditioned is determined by $\mathrm{co}(u)$, i.e., $\mathcal{O}(\exp(w))$, where width w is equal to $|\mathrm{co}(u)|$ (Definition 16). It follows that inference time complexity is $\mathcal{O}(n \exp(w))$, where $n = |S| = |V|$ and w is the tree-width of $\mathcal{T}$. □

Corollary 1. *The compositional approach offers inference time complexity $\mathcal{O}(n \exp(w))$, where $n = |X_V|$ and w is the tree-width of Bayesian network $(\mathcal{G}, P)$, with $\mathcal{G} = (V, E)$, if the symbolic target representation chosen for compilation is a subset of d-DNNF.*

Proof. Note that there exists a composition tree that is equivalent to a junction tree. There is therefore a composition tree with a tree-width that is bounded by the size of the largest set in the junction tree. Since this largest set in the junction tree defines the tree-width of the Bayesian network, the inference time complexity follows immediately from Theorem 1. □

This complexity result is comparable to mainstream algorithms based on variable elimination, clustering and conditioning (exponential in the network tree-width and linear in its size) [16]. However, space requirements can be reduced significantly due to partitioning. Moreover, local structure can still be exploited like in the monlithic approach. For instance, causal independence is defined within families [26]. Since the parent nodes always reside in the same partition as the child (possibly in the context, as duplicates of nodes from another partition), any decision diagram representation can still fully exploit this structure just like in the monolithic approach, yielding an exponential reduction in the number of parents [45]. Any exploitation of structure that would be present in the monolithic (unpartitioned) representation is however not necessarily present in the message graph. In the worst case, our inference algorithm might visit a partition more often than necessary, given the presence of e.g. causal independence in the context variables. In future versions of our approach, we intend to remedy this by performing inference in higher partitions in lock step with the lower partitions, synchronizing whenever a

(shared) context variable is encountered. This way the inference algorithm would also fully benefit from the causal inference. A downside would be that the order of context variables would become fixed across partitions, whereas our current approach can exploit any order within a partition to further reduce its representation size.

6. Optimizing the framework

The bottleneck of the WMC approach is compilation, however, the compositional approach reduces both compilation and inference cost if the partitioning, subproblem- and compilation orders are selected with care. The size of target representations depends on the ordering that is imposed on the variables during compilation. Subsequently, the ordering in which compiled subproblems are composed influences inference cost, which in turn is dictated by the partitioning.

6.1. Finding a partitioning

Partitioning influences both compilation and inference cost, and obtaining one of high quality is therefore crucial. Exhaustively trying to find an optimal partitioning with the corresponding variable orderings is intractable. A good partitioning is found by using a scoring function that provides implications to its quality, based on the size of the resulting target representation.

Any optimization technique may be used to find a partitioning. *Simulated annealing* was chosen to perform this task, although there are many alternatives (e.g., local search, evolutionary algorithms, etc). The scoring function to be introduced for composition-trees (Section 6.2) is combined with a metric for minimizing cutset size. They are combined using a harmonic mean. Subproblems with the largest score are recursively split until a desirable overall bound is reached. Dealing with *connected components* in the moralized graph of the BN is beneficial, as produced symbolic representations are typically much smaller.

6.2. Finding a composition-tree

Let $\mathcal{C} = (\mathcal{B}, S, \mathcal{T}, \mathcal{U})$ be a composition problem. A composition-tree is found by providing Algorithm 1 with an ordering on $\mathcal{S}$. The context of the nodes in a composition-tree $\mathcal{T} = (S, \mathcal{H}), \mathcal{H} = (U, F)$, provides implications to the size of the message graph $\mathcal{U}$. The message graph is the most dominant factor in determining inference cost. To find a good composition-tree, any optimization technique may be used to find an ordering that minimizes Equation (17).

$$\sum_{v \in U} \mathsf{w}(v), \tag{17}$$

where width w is provided by Definition 16.

6.3. Finding a compilation ordering

The same principles of tree-width and context that apply to composition-trees (Definition 11) can be applied to compilation orderings. Just as for composition-trees, a directed tree can be created from any (partial) ordering.

Definition 17 *(Proper compilation ordering).* Let $\mathcal{B} = (\mathcal{G}, P)$ be a BN with DAG $\mathcal{G} = (V, E)$. A partial order $(X_V, \prec)$ is said to be *proper* if $X_{\mathsf{fa}(u)} \cap X_{\mathsf{fa}(v)} \neq \emptyset$ then $X_u \prec X_v$ or $X_v \prec X_u$, with $u, v \in V$.

Definition 18 *(Compilation-tree).* Let $(X_V, \prec)$ be a proper partial ordering for BN $\mathcal{B} = (\mathcal{G}, P)$, $\mathcal{G} = (V, E)$. A *compilation-tree* $\mathcal{K} = (X_V, \mathcal{H})$ is a directed rooted tree $\mathcal{H} = (U, F)$, with nodes U and edges $F \subseteq U \times U$, such that there is a one-to-one correspondence between nodes U and nodes V, and there is a directed path from u to v iff $X_u \prec X_v$, for $u, v \in U$.

Definition 19 *(Compilation-tree context).* Let $\mathcal{K} = (X_V, \mathcal{H})$ be a compilation-tree with graph $\mathcal{H} = (U, F)$ for BN $\mathcal{B} = (\mathcal{G}, P)$, $\mathcal{G} = (V, E)$. The *context* of node $v \in U$ is defined as:

$$\mathsf{co}(v) \triangleq \left(X_{\mathsf{fa}_{\mathcal{G}}(v)} \cup \bigcup_{u \in \mathsf{ch}_{\mathcal{H}}(v)} \mathsf{co}(u) \right) \cap \bigcup_{u \in \mathsf{an}_{\mathcal{H}}(v)} X_{\mathsf{fa}_{\mathcal{G}}(u)}. \tag{18}$$

Note that co is differently defined for composition- and compilation-trees. Context is also used for compilation-trees. In fact, the size of the largest context set is often referred to as the tree-width. Algorithm 4 is a work efficient implementation of Definition 19, with near linear time complexity in the number of (subproblem) BN variables. It takes as input a sequence of BN variables and induces a partial ordering based on BN connectivity. A compilation-tree is induced by the partial ordering that is consistent with Definition 18. The number of root nodes it returns is equal to the number of connected components in the (subproblem) BN. Note that the compilation-tree is guaranteed to be induced by a total ordering, creating

Algorithm 4 Create a compilation-tree.

CreateCompilationTree($\mathcal{B}, \mathcal{S}_i, \sigma, b$)

input: BN $\mathcal{B} = (\mathcal{G}, P)$, subproblem $\mathcal{S}_i = (Z_i, \mathcal{G}_i, \psi_i)$, permutation function σ over variables X_{Z_i} and Boolean b
output: Compilation-tree $(X_{Z_i}, \mathcal{H})$, $\mathcal{H} = (U, F)$.

$Z = Z_i$
$U = \{\}$, $F = \{\}$, $W = \{\}$
for $j = |X_{Z_i}|$ **to** 1
  $v = \sigma(j)$
  $U = U \cup \{v\}$
  $F = F \cup \{(v, u) | u \in W, X_v \in \mathsf{co}(u) \vee b\}$
  $\mathcal{H} = (U, F)$
  $W = (W \setminus \mathsf{ch}_{\mathcal{H}}(v)) \cup \{v\}$
  $Z = Z \setminus \{v\}$
  $\mathsf{co}(v) = \left(X_{\mathsf{fa}_{\mathcal{G}}(v)} \cup \bigcup_{u \in \mathsf{ch}_{\mathcal{H}}(v)} \mathsf{co}(u) \right) \cap \bigcup_{z \in Z} X_{\mathsf{fa}_{\mathcal{G}}(z)}$.

return $(X_{Z_i}, (U, F))$

Table 2
Computing scoring functions for Example 13.

			v_1	v_2
MDD	Number of nodes v_i	$\prod_{Y \in \mathsf{co}(v_i)} \mathsf{car}(Y)$	1	3
	Total number of nodes	$\sum_{v \in \{v_1, \ldots, v_i\}} \prod_{Y \in \mathsf{co}(v)} \mathsf{car}(Y)$ (Eq. (19))	1	<u>4</u>
WPBDD	Number of nodes v_i	$\prod_{Y \in \mathsf{co}(v_i) \cup X_{v_i}} \mathsf{car}(Y)$	3	6
	Total number of nodes	$\sum_{v \in \{v_1, \ldots, v_i\}} \prod_{Y \in \mathsf{co}(v) \cup X_v} \mathsf{car}(Y)$	3	<u>9</u>

a chain, if the final argument to CreateCompilationTree is **1**. This is useful for those target representations that are induced by total orderings, such as OBDDs.

Given a subproblem $\mathcal{S}_i = (Z_i, \mathcal{G}_i, \psi_i)$, a compilation-tree is created by providing a sequence of X_{Z_i} (as permutation function σ) to Algorithm 4. The context of nodes in a compilation-tree $\mathcal{K} = (X_{Z_i}, \mathcal{J})$, $\mathcal{J} = (W, K)$, provides implications to the size of symbolic representations to be produced through compilation. A permutation is transformed into a compilation-tree with Algorithm 4 and scored by Equation (19). This score is used to compare a permutation to other permutations. Any optimization technique may be used to traverse the search space of all possible permutations of X_{Z_i} to find a good compilation-tree by minimizing Equation (19). In our experiments, we have used simulated annealing to achieve this.

$$\sum_{v \in W} \mathsf{car}(\mathsf{co}(v)). \tag{19}$$

To minimize the chance of integer overflow, the product term can be substituted with $\sum_{X \in \mathsf{co}(v)} \log_{10}(\mathsf{car}(X))$, which is proportionally equivalent. Actually, Equation (19) can be tailored to behave as the upper bound for most of the (binary) decision diagram types used as target representation in recent work. For WPBDDs, the context variables $\mathsf{co}(v)$ of node v in the compilation-tree would simply have to be unioned with $\{X_v\}$, and Equation (19) recomputed. This is demonstrated in Example 13.

Example 13. Consider the BN of Example 1. Assuming no local structure, Fig. 14 shows a multi-valued decision diagram, given ordering $A \prec B$ (Fig. 14b), and a WPBDD given induced ordering $a_1 \prec a_2 \prec a_3 \prec b_1 \prec b_2$. It shows how Equation (19) relates the upper bound for MDDs and WPBDDs, which is computed in Table 2. The underlined numbers it contains, computed by Equation (19), are equal to the number of MDD and WPBDD nodes.

Equation (19) is computed for monolithic representations as well as individual subproblems.

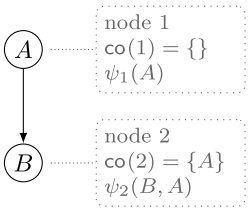
7. Related work

Probabilistic inference is a difficult problem in Artificial Intelligence [11]. The use of logic remains to play an important role in AI [17,27], even though there are successful techniques that do not require modeling or reasoning such as neural

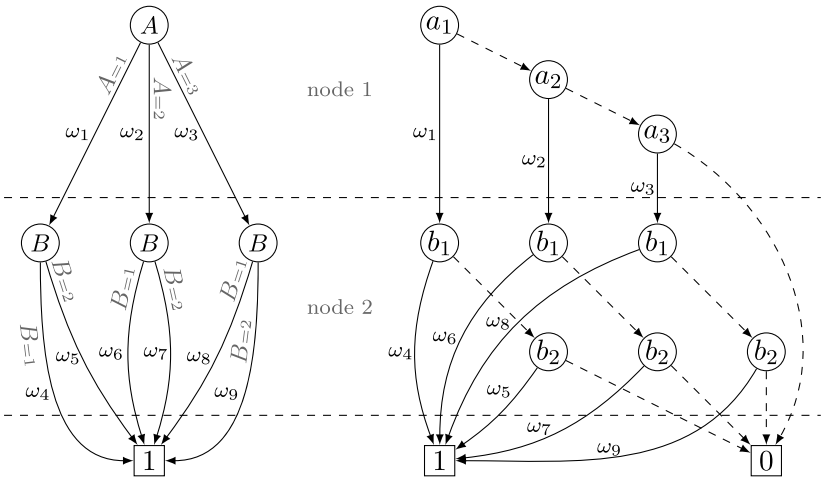
$P(A=1)$	$P(A=2)$	$P(A=3)$
ω_1	ω_2	ω_3

A	$P(B=1\|A)$	$P(B=2\|A)$
1	ω_4	ω_5
2	ω_6	ω_7
3	ω_8	ω_9

(a) CPTs.



(b) Compilation-tree.



(b) MDD. **(c)** WPBDD.

Fig. 14. Showing the relation between context and representation size for Example 13.

networks [9]. Inference by Weight Model Counting (WMC) is a logical approach that has been very successful in solving reasoning problems in the past decade [6–8,14]. WMC is just one out of a collection of algebraic approaches that also includes, but is not limited to, Symbolic Probabilistic Inference (SPI) [43], Recursive Conditioning (RC) [16], the bucket elimination algorithm [19] and sum-product networks [46].

These approaches essentially perform inference in the same way, by (first) trying to find the most concise factorization of the function that computes marginal probabilities. For the sum-product approach, this factorization is recorded as a sum-product network [46], for the WMC approach it can be recorded into a large variety of (target) representations. The process of obtaining such a concise representation is also referred to as Knowledge Compilation (KC) [18]. KC is acknowledged as a challenging approach that makes many practical reasoning problems tractable [33].

Numerous target representations have been used to concisely model probability distributions, providing more concise factorizations than BNs in the presence of local structure. Examples include Deterministic Decomposable Negation Normal Form (d-DNNF) [7], Sentential Decision Diagrams (SDD) [8], Probabilistic SDDs (PSDD) [32,44], Ordered Binary Decision Diagrams (OBDD) [40], Zero-suppressed BDDs (ZBDD) [35], And/Or Multi-Valued Decision Diagrams (AOMDDs) [34], Probabilistic decision graphs [29], Weighted Positive BDDs (WPBDD) [14], Multi-Valued Decision Diagrams [1], Algebraic Decision Diagrams (ADD) [21], among others. Unfortunately, the cost of compiling a BN into a symbolic representation remains a bottleneck. However, learning its structure seems even more costly [32].

We tackle the issues that related work has revealed concerning compilation cost using a framework that relies on partitioning [15], also leading to possibilities for parallel computation [12,13]. The proposed approach bears resemblance to RC, in that inference is also solved as a set of subproblems [16]. The notion of conditioning out the variables in a subproblem that it shares with ancestor subproblems, is similar. Contextual information being passed upward in a dtree corresponds to that in a composition-tree. Both lead to tree decompositions, that reduce the connectivity of the network. It is therefore no coincidence that inference time complexity is comparable to RC (exponential in the network tree-width and linear in its size), and therefore also comparable to mainstream algorithms based on variable elimination and clustering [16].

However, there are key differences. RC decomposes a network into smaller subnetworks that are then solved recursively using the same method until single-node subnetworks can be solved. The disintegration of the network into its smallest

(single-node) components is not required with CWMC, and arguably disadvantageous even. It is encouraged to allow state-of-the-art compilers to play to their strength, i.e., finding a concise representation (by exploiting local structure) over an as large as possible subproblem. Also, each execution of RC answers only a single query, while CWMC results in a compact structure that can be amortized over potentially a very large number of queries.

The work in this paper can also be understood in the framework of structured message passing [24] (SMP). In SMP, a so-called structured cluster graph is introduced which is a very generic graph representation where nodes and edges are associated with parametric functions. The authors show that message passing algorithms can in principle be applied to these graphs. More concretely, the authors introduce an algorithm that exploits sparse tables or algebraic decision diagrams and introduce context-specific independence and determinism in its messages using approximate methods. The algorithm in this paper is quite different: the main purpose of the proposed algorithm is to enable *exact* inference using structured cluster graphs. The surprising observation is that even with large cluster graphs, significant performance gains can be reached, without approximating the posteriors. In contrast, for SMP algorithm, the variance increases if the cluster size increases.

Partitioning of propositional theories has also been exploited in model checking to speed up and distributed symbolic approaches using decision diagrams [25,39,42]. These approaches rely on predefined window functions for partitioning. The CWMC approach, on the other hand, performs the partitioning prior to compilation using a syntactic approach informed by the structure of the BN. While this allows fewer partitions than windowing functions would, it also makes the approach more compositional, allowing for example different variable orderings in each partitioned subproblem.

The most general language that is known to support efficient model counting is d-DNNF [18]. Although it is currently not transparent how to adapt d-DNNF to the proposed framework, a strict subset, called decision-DNNF, can be converted into an equivalent Free Binary Decision Diagram (FBDD) with only a quasi-polynomial increase in representation size [2]. Thus, the proposed method applies to representations that are less or equally succinct as decision-DNNF, e.g., SDDs, OBDDs, ZBDDs, WPBDDs.

WMC has recently been extended to Weighted Model Integration (WMI), which can be used to solve probabilistic reasoning problems that involve both discrete and continuous probability distributions [37]. It is shown that standard knowledge compilation techniques apply to WMI, leading to exact and approximate solvers [20,38]. Based on this finding, we argue that the proposed partitioning technique in this article can extend WMI for representations that are less or equally succinct as d-DNNF when dealing with non-linear real arithmetics. The weighted model integral for a non-factorizable weight function is obtained by adding up the weighted model integrals for the factorizable weight functions into which the problem decomposes. Every time knowledge compilation is applied within this framework, partitioning can also be applied. We leave this for future work.

8. Empirical results

Several publicly available Bayesian networks[2] are used to evaluate the performance of compilation and inference while employing the proposed compositional approach. All experiments ran on a system with AMD Opteron 6376 processors, with 500+ Gb of RAM.

8.1. Compilation

The main goal of the following experiments is to show the improvement in compilation cost provided by the CWMC framework. However, we also provide a compilation comparison to SDDs,[3] OBDDs[4] and d-DNNFs.[5] CWMC is orthogonal to the language used. Observed improvements therefore have implications for the mentioned languages, because they can also be used in the CWMC framework.

A compiler was implemented that supports compilation to partitioned WPBDDs (induced by chain- or total orders) and t-WPBDDs (induced by tree- or partial orders).[6] Table 3 contains compilation runtimes for both aforementioned representations. Monolithic compilation as well as partitioned compilation was reviewed for multiple partitionings.

The compilation experiment was set up as a head-to-head procedure that invokes the compilers of respective representations, i.e., the same steps are performed in the same order to produce each representation to ensure a fair comparison. Partitioned (t-)WPBDDs, (t-)WPBDDs, SDDs and OBDDs are produced by (1) encoding each BN as a Boolean formula (Section 2.3.2); (2) one (total) variable ordering is created per BN using the popular minimum-fill greedy heuristic. Although this heuristic is proven to produce reasonable approximations, we have further optimized the produced ordering using simulated annealing based on the methods described in Section 6. Orders are relaxed into partial orders based on BN constraints for t-WPBDDs (Algorithm 4); (3) the same compilation ordering is used for all languages. Firstly, a respective representation is created for each individual CPT. Secondly, CPT representations are conjoined until the final representation is obtained. This compilation type is referred to as bottom-up [8]. Compilation runtimes reported in Table 3 only concern step 3.

[2] BN collection is available at https://github.com/gisodal/wmc/tree/master/data/net.
[3] The SDD compiler is available at http://reasoning.cs.ucla.edu/sdd.
[4] The CUDD compiler (for OBDDs) is available at http://vlsi.colorado.edu/~fabio.
[5] The ACE compiler (for d-DNNFs) is available at http://reasoning.cs.ucla.edu/ace.
[6] The WPBDD compiler/model counter is available at https://github.com/gisodal/wmc.

Table 3
Compilation runtime (milliseconds), where $|\mathcal{A}(X)|$ are the number encoding variables for BN variables X, - implies compilation failure by exceeding 15 minutes or 500Gb of RAM memory.

Bayesian Network	$\|at(X)\|$	4 subproblems t-WPBDD	2 subproblems t-WPBDD	t-WPBDD	4 subproblems WPBDD	2 subproblems WPBDD	WPBDD	OBDD	SDD	d-DNNF
sachs	24	0.148	0.115	**0.100**	0.790	0.471	0.286	1.932	29.119	92.179
student farm	25	0.117	**0.101**	0.106	0.816	0.519	0.335	1.403	4.646	118.641
printer ts	58	0.230	0.216	**0.198**	1.015	0.729	0.604	1.757	6.628	97.956
boblo	60	0.213	**0.198**	0.213	0.929	0.607	0.494	3.792	27.920	118.202
child	60	**0.195**	0.199	0.331	0.903	0.605	0.563	4.564	96.344	117.620
insurance	89	**0.494**	2.187	20.365	1.292	2.395	27.508	267.967	12337.980	680.771
weeduk	90	18.415	18.973	6.091	98.887	44.697	**4.206**	429.110	-	3472.012
alarm	105	**0.407**	0.474	0.467	1.183	1.052	1.040	10.085	400.158	157.163
water	116	**5.185**	17.134	1635.935	8.002	23.789	1060.444	16034.149	-	1009.578
powerplant	120	**0.268**	0.281	0.361	1.022	0.819	0.764	9.409	119.856	159.193
carpo	122	0.426	**0.407**	0.420	1.401	1.159	1.200	13.910	119.122	137.955
win95pts	152	**0.874**	1.199	1.386	2.548	3.670	5.784	193.919	902.473	173.762
hepar2	162	**1.444**	1.684	1.567	2.344	2.628	4.722	414.316	31119.984	287.980
fungiuk	165	22.186	28.469	45.559	**11.098**	15.086	322.238	1667.940	-	12193.593
hailfinder	223	**1.061**	2.382	3.748	2.136	3.757	16.494	422.270	14350.353	354.151
3nt	228	**0.696**	0.727	2.397	1.892	2.161	9.393	344.902	4259.798	424.939
4sp	246	0.849	**0.819**	5.090	1.973	2.436	20.175	991.545	7041.476	573.015
barley	421	611.830	9294.794	23290.743	**340.912**	854.859	248800.832	-	-	-
mainuk	421	584.409	8456.920	23443.483	**322.308**	872.767	35236.875	-	-	-
andes	440	**3.267**	5.159	224.648	17.217	130.747	-	-	-	7785.916
pathfinder	520	**17.279**	17.506	18.057	48.054	49.029	62.378	22741.434	137591.643	2813.821
mildew	616	42.611	43.675	576.852	43.677	**38.928**	830.392	244920.444	-	885305.099
munin1	992	**11.929**	13797.173	53899.548	131.640	-	-	-	-	-
pigs	1323	**4.444**	13.538	348.872	46949.737	-	-	-	-	20623.511
link	1793	**174.897**	414.150	19412.863	-	-	-	-	-	-
diabetes	4682	**1297.221**	2024.698	2622.924	-	-	-	-	-	-
munin2	5376	**96.809**	263.800	926.789	-	-	-	-	-	235544.805
munin3	5601	**52.350**	1046.767	1088.710	-	-	-	-	-	102338.718
munin4	5645	**718.358**	1705.222	2565.931	-	-	-	-	-	162054.255
munin	5651	**1407.531**	2143.027	2360.196	-	-	-	-	-	161133.160

Unfortunately, ACE (producing d-DNNFs) could not be conformed to the compilation procedure, as it does not have a library interface. It also does not accept a compilation ordering as input. ACE reports several runtimes of various operations, e.g., encoding-, initialization time, etc. Table 3 solely reports ACE's compile time. Several approaches were used to optimize SDD compile time. An SDD is compiled using an ordering type called a vtree [8]. SDDs were compiled using a balanced vtree. These were induced by orders optimized using multiple different scoring functions. Right-aligned vtrees produced SDDs that have the same number of operators as OBDDs (after having translated n-ary operators to n-1 binary operators). This is known [8]. Although these representations are equal, compilation times were consistently higher for SDDs than those for OBDDs and are therefore omitted. Note that better compilation times have been reported using different vtrees [8], but proposed methods remain a significant improvement.

Observe in Table 3 that partitioned compilation often leads to speedups of multiple orders of magnitude and progressively improves when increasing the number of subproblems, especially for larger BNs. For smaller BNs, this is not always the case. The explanation lies with the size of produced representations reported in Table 4. Partitioned representations can increase in size because cutset variables are present in multiple subproblems. This redundancy only leads to an increase for small BNs in practice. Generally, the size reduces and does so for two reasons. Local structure is better exploited, and constraints that encode edges in a partitioning's cutset are not represented in the compiled structure. These constraints are encoded by the message graph, which is a dynamically inferred structure used during inference (see Section 3.2). Note that the representation sizes of d-DNNFs and t-WBDDs are quite similar, in particular for the large BNs. However, compiling t-WPBDDs is around two orders of magnitude faster and improves even further by orders of magnitude with partitioning. As a result, there are multiple instances where a BN can now successfully be compiled that previously could not be given resource constraints.

8.2. Inference

Algorithm 2 was implemented to perform probabilistic inference as part of the compositional approach.[6] A comparison is made with model counter ACE.[5] The inference experiment was setup as a head-to-head procedure, where each inference method computes a particular query. Queries are created randomly, i.e., with a random number of observed variables and a random configuration. A query is created and fed to each method. This process repeats for 30 minutes per network. In the end, each method has computed the same set of queries. Table 5 reports the average runtime per query.

For a fair comparison, none of the methods were optimized using conditional independence, e.g., by only traversing conditionally dependent subproblems. Also, the number of subproblems is no longer fixed as with compilation. In addition, the WMC methods do not take advantage of caching, such that every query requires to be completely recomputed in order

Table 4
Representation size (number of binary operators), where $|\mathcal{A}(X)|$ is the number encoding variables for BN variables X, - implies compilation failure by exceeding 15 minutes or 500Gb of RAM memory.

Bayesian Network	$\|at(X)\|$	4 subproblems t-WPBDD	2 subproblems t-WPBDD	t-WPBDD	4 subproblems WPBDD	2 subproblems WPBDD	WPBDD	OBDD	SDD	d-DNNF
sachs	24	95	77	82	98	77	**73**	534	4437	510
student farm	25	**41**	43	80	**41**	43	76	366	1879	221
printer ts	58	33	33	33	32	30	**29**	118	671	117
boblo	60	97	**93**	160	152	136	206	1439	6429	371
child	60	**117**	151	267	121	159	400	2814	16664	1069
insurance	89	**390**	1698	28049	421	1439	27004	236560	4508128	31599
weeduk	90	**1526**	1531	1540	5866	3007	2151	37272	-	15001
alarm	105	321	449	470	**235**	519	693	4620	64797	1373
water	116	2973	18891	339581	**2103**	26037	237117	9498498	-	21297
powerplant	120	190	239	244	**164**	296	504	3899	44511	1285
carpo	122	**222**	237	301	260	311	909	4836	19209	1049
win95pts	152	**525**	1180	1582	965	2847	4558	21523	154946	2577
hepar2	162	**1215**	1466	1496	1379	2596	7675	49892	417093	6785
fungiuk	165	**2003**	4216	6563	6769	8202	9289	187140	-	48600
hailfinder	223	**973**	1587	3169	1261	3731	7426	116514	3098190	6489
3nt	228	**501**	575	3088	697	1144	9130	73004	1581601	5626
4sp	246	**617**	663	7421	692	1529	21904	147678	1824426	8245
barley	421	75595	560856	18156823	**70507**	440698	104535392	-	-	-
mainuk	421	**74209**	532850	17813603	90106	429885	13408167	-	-	-
andes	440	**2896**	5609	378413	13969	54950	-	-	-	480184
pathfinder	520	**3689**	3747	3848	28842	30191	35471	1151753	3697971	10580
mildew	616	**6701**	6782	205617	7674	12694	424448	11610911	-	734902
munin1	992	**12185**	3469305	22270466	49784	-	-	-	-	-
pigs	1323	**4625**	24855	528055	30066054	-	-	-	-	521776
link	1793	**194891**	541288	27704409	-	-	-	-	-	-
diabetes	4682	**1061818**	2139899	3202034	-	-	-	-	-	-
munin2	5376	**80606**	227125	1130577	-	-	-	-	-	1348670
munin3	5601	**52076**	985793	1114785	-	-	-	-	-	711667
munin4	5645	**440412**	1236879	1461146	-	-	-	-	-	1379331
munin	5651	**942741**	1275268	1435672	-	-	-	-	-	1410553

Table 5
Inference runtime averaged per query (milliseconds), where - implies inference failure by exceeding 15 seconds, or compilation failure (see Table 3).

Bayesian Network	Partitioned t-WPBDD	t-WPBDD	d-DNNF
sachs	0.023	**0.011**	2.975
student farm	0.035	**0.016**	2.813
printer ts	0.007	**0.006**	2.852
boblo	0.054	**0.034**	3.713
child	0.345	**0.036**	5.695
insurance	7.486	**1.874**	36.884
weeduk	0.607	**0.262**	30.908
alarm	0.187	**0.115**	6.513
water	**25.176**	74.135	33.512
powerplant	**0.025**	0.032	6.249
carpo	0.138	**0.037**	5.739
win95pts	**0.371**	0.635	9.680
hepar2	**0.247**	1.133	18.659
fungiuk	18.822	**5.290**	42.814
hailfinder	25.137	**1.747**	19.618
3nt	18.411	**8.250**	21.559
4sp	5.748	**1.277**	30.043
barley	**1399.542**	1798.278	-
mainuk	**1377.117**	1782.512	-
andes	178.610	185.205	**144.691**
pathfinder	5.394	**0.639**	30.686
mildew	351.788	552.496	**208.582**
munin1	7183.857	**6836.045**	-
pigs	248.866	**70.266**	179.088
link	**9893.431**	-	-
diabetes	968.839	**618.687**	-
munin2	**107.788**	207.768	384.055
munin3	828.535	**140.398**	263.751
munin4	**280.275**	318.687	402.651
munin	377.117	**302.675**	416.733

to get as close to the core method as possible. The time a method can take to perform inference is limited to 15 seconds, because CWMC is able to succeed within this timeframe for both compilation and inference combined with all networks.

It is known that partitioning a problem into $m+1$ as opposed to m subproblem can lead to exponential reductions [36]. An attempt is therefore made via simulated annealing to automatically find a good partitioning. A BN partitioning is guaranteed to produce m connected components by creating a spanning tree of the BN and removing $m-c$ edges, where c is the number of connected components in the BN without partitioning, and $m \geq c$. The connected components in the spanning tree induce connected components in the BN. A partitioned t-WPBDD is guaranteed to consist of at least 2 and at most 12 connected components during the experimentation.

Partitioned compilation has lead to speedups of orders of magnitude, even for a very limited number of subproblems like 2 and 4. Table 5 shows that inference cost is also reduced in multiple instances when employing the compositional approach. Any increase or decrease in inference cost is closely related to the quality of the partitioning and the composition-tree used to induce the message graph. Any constraint not present in the compiled subproblems given its partitioning must be inferred by the message graph. A small overhead is incurred every time a switch is made from subproblem to subproblem. However, in practice we see reductions regardless of this overhead. Data locality plays a major role in performance. Quite simply, there are more cache misses for monolithic representations than there are for composed representations. This was confirmed using a profiler. Not only are composed representations typically much smaller than monolithic ones, caching is done per subproblem, which are even smaller still. From a theoretical perspective, this is further aided by the improved ability to capture local structure in subproblems, leading to more concise representations for critical parts of the BN.

Inference cost can be further improved in the future. For instance, conditional independence can be used to dynamically prune the composed representation. This leads to significant reductions in inference cost for large sets of probabilistic queries. Also, working with subproblems provides opportunities for parallel processing [13]. In addition, our focus regarded the validity and value of compositional compilation and inference. Improving the quality of partitioning will result in overall reductions for both compilation and inference.

9. Conclusion

Weighted Model Counting (WMC) has been recognized as a state-of-the-art technique for exact probabilistic inference. It improves further the factorization of a BN by exploiting local structure ([8,14,40]). However, it requires a computationally intensive compilation task in order to yield an optimized representation upon which efficient inference can be performed. As a result, this method cannot be used in many real-world domains. We have proposed a framework, called Compositional Weighted Model Counting (CWMC), that extends existing state-of-the-art compilers and model counters to tackle computation costs. These include the SDD, CUDD and WPBDD compilers, and the ACE and CACHET model counters.

Several advantages emerge when using CWMC. From a theoretical point of view, CMWC allows for a flexible representation because subproblems are compiled locally. For example, if subproblems make use of a representation where a variable ordering is imposed, then this ordering may vary between subproblems. As a consequence, this provides more fine-grained control to exploit BN structure and topology. Furthermore, since subproblems are compiled locally, the complexity of reasoning can be kept under control more easily by exploiting ideas from Boolean factorization where exponential reductions in runtime have been observed [36]. While the worse-case inference complexity of CWMC is similar to existing Bayesian network inference methods, i.e., $\mathcal{O}(n \exp(w))$, where n is the number of variables and w is the tree-width of the network, we observe in experiments that inference can be several magnitudes more efficient using the proposed compositional approach compared to standard WMC methods.

The experimental evaluation reveals several benefits: (1) the compilation cost is drastically reduced while using only a limited number of partitions, (2) the representations obtained are much smaller, thus reducing resource requirements, and (3) inference cost has also decreased in several instances. Future work will include refining methods to obtain better orderings and partitionings to improve these results, and reducing inference cost by exploiting conditional independence and parallel processing.

The reduction in compilation cost can either be reinvested in a search for even more concise representations, or can inspire to handle much larger Bayesian networks than previously possible. Additionally, WMC methods have only been scarcely applied when dealing with dynamic data, as any changes would require the recompilation of the entire representation. Using the proposed partitioning approach, this would not be necessary. Merely the partition(s) that represent the affected part(s) of the BN would have to be recompiled. With the proposed work we strive to overcome (at least in part) the current limitations of WMC based methods, and cater to the growing need for algorithms that can deal with very big Bayesian networks or Bayesian networks that are based on dynamic data.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] J. Amilhastre, H. Fargier, A. Niveau, C. Pralet, Compiling CSPs: a complexity map of (non-deterministic) multivalued decision diagrams, Int. J. Artif. Intell. Tools 23 (2014) 146–166.
[2] P. Beame, J. Li, S. Roy, D. Suciu, Exact model counting of query expressions: limitations of propositional methods, ACM Trans. Database Syst. 42 (1) (2017) 1–46.
[3] B. Bollig, I. Wegener, Improving the variable ordering of OBDDs is NP-complete, IEEE Trans. Comput. 45 (1996) 993–1002.
[4] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: International Conference on Uncertainty in Artificial Intelligence, 1996, pp. 115–123.
[5] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. 100 (1986) 677–691.
[6] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artif. Intell. 172 (2008) 772–799.
[7] M. Chavira, A. Darwiche, M. Jaeger, Compiling relational Bayesian networks for exact inference, Int. J. Approx. Reason. 42 (2006) 4–20.
[8] A. Choi, D. Kisa, A. Darwiche, Compiling probabilistic graphical models using sentential decision diagrams, in: European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty, 2013, pp. 121–132.
[9] A. Choi, R. Wang, A. Darwiche, On the relative expressiveness of Bayesian and neural networks, Int. J. Approx. Reason. 113 (2019) 303–323.
[10] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.J. Spiegelhalter, Probabilistic Networks and Expert Systems, 1999.
[11] P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, Artif. Intell. 60 (1993) 141–153.
[12] G.H. Dal, W.A. Kosters, F.W. Takes, Fast diameter computation of large sparse graphs using GPUs, in: International Conference on Parallel, Distributed and Network-Based Processing, 2014, pp. 632–639.
[13] G.H. Dal, A.W. Laarman, P.J.F. Lucas, Parallel probabilistic inference by weighted model counting, in: International Conference on Probabilistic Graphical Models, 2018, pp. 97–108.
[14] G.H. Dal, P.J.F. Lucas, Weighted positive binary decision diagrams for exact probabilistic inference, Int. J. Approx. Reason. 90 (2017) 411–432.
[15] G.H. Dal, S. Michels, P.J.F. Lucas, Reducing the cost of probabilistic knowledge compilation, J. Mach. Learn. Res. 73 (2017) 141–152.
[16] A. Darwiche, Recursive conditioning, Artif. Intell. 126 (2001) 5–41.
[17] A. Darwiche, Three modern roles for logic in AI, in: Symposium on Principles of Database Systems, 2020, pp. 229–243.
[18] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artif. Intell. Res. 17 (2002) 229–264.
[19] R. Dechter, Bucket elimination: a unifying framework for probabilistic inference, in: Learning in Graphical Models, 1998, pp. 75–104.
[20] P.Z. Dos Martires, A. Dries, L. De Raedt, Exact and approximate weighted model integration with probability density functions using knowledge compilation, in: International Conference on Artificial Intelligence, vol. 33, 2019, pp. 7825–7833.
[21] J.M. Dudek, V. Phan, M.Y. Vardi, ADDMC: weighted model counting with algebraic decision diagrams, in: International Conference on Artificial Intelligence, 2020, pp. 1468–1476.
[22] N. Friedman, M. Goldszmidt, Learning Bayesian networks with local structure, in: Learning in Graphical Models, 1998, pp. 421–459.
[23] M.A. Genesereth, N.J. Nilsson, Logical Foundation of Artificial Intelligence, 1987.
[24] V. Gogate, P. Domingos, Structured message passing, in: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, 2013, pp. 252–261.
[25] O. Grumberg, T. Heyman, A. Schuster, A work-efficient distributed algorithm for reachability analysis, Form. Methods Syst. Des. 29 (2) (2006) 157–175.
[26] D. Heckerman, J. Breese, Causal independence for probabilistic assessment and inference using Bayesian networks, IEEE Trans. Syst. Man Cybern. 26 (1996) 826–831.
[27] T.C. Henderson, R. Simmons, B. Serbinowski, M. Cline, D. Sacharny, X. Fan, A. Mitiche, Probabilistic sentence satisfiability: an approach to PSAT, Artif. Intell. 278 (2020) 103–118.
[28] A. Hommersom, P.J.F. Lucas, M. Velikova, G.H. Dal, MoSHCA - my mobile and smart health care assistant, in: International Conference on e-Health Networking, Applications and Services, 2013, pp. 188–192.
[29] M. Jaeger, Probabilistic decision graphs, combining verification and AI techniques for probabilistic inference, Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 12 (2004) 19–42.
[30] S.L. Lauritzen, Graphical Models, 1996.
[31] W. Li, P. Poupart, P. van Beek, Exploiting structure in weighted model counting approaches to probabilistic inference, J. Artif. Intell. Res. 40 (2011) 729–765.
[32] Y. Liang, J. Bekker, G. Van den Broeck, Learning the structure of probabilistic sentential decision diagrams, in: International Conference on Uncertainty in Artificial Intelligence, 2017.
[33] P. Marquis, Compile!, in: International Conference on Artificial Intelligence, 2015, pp. 4112–4118.
[34] R. Mateescu, R. Dechter, R. Marinescu, AND/OR multi-valued decision diagrams (AOMDDs) for graphical models, J. Artif. Intell. Res. 33 (2008) 465–519.
[35] S.-i. Minato, K. Satoh, T. Sato, Compiling Bayesian networks by symbolic probability calculation based on zero-suppressed BDDs, in: International Joint Conference on Artificial Intelligence, 2007, pp. 2550–2555.
[36] A. Mintz, M.C. Golumbic, Factoring Boolean functions using graph partitioning, Discrete Appl. Math. 149 (2005) 131–153.
[37] P. Morettin, A. Passerini, R. Sebastiani, Advanced smt techniques for weighted model integration, Artif. Intell. 275 (2019) 1–27.
[38] P. Morettin, S. Kolb, S. Teso, A. Passerini, Learning weighted model integration distributions, in: International Conference on Artificial Intelligence, 2020, pp. 5224–5231.
[39] A. Narayan, J. Jain, M. Fujita, A. Sangiovanni-Vincentelli, Partitioned ROBDDs-a compact, canonical and efficiently manipulable representation for Boolean functions, in: Proceedings of International Conference on Computer Aided Design, 1996, pp. 547–554.
[40] T.D. Nielsen, P.-H. Wuillemin, F.V. Jensen, U. Kjaerulff, Using ROBDDs for inference in Bayesian networks with troubleshooting as an example, in: International Conference on Uncertainty in Artificial Intelligence, 2000, pp. 426–435.
[41] R. Paredes, L. Dueñas-Osorio, K. Meel, M. Vardi, A weighted model counting approach for critical infrastructure reliability, in: International Conference on Applications of Statistics and Probability in Civil Engineering, 2019.
[42] D. Sahoo, S. Iyer, J. Jain, C. Stangier, A. Narayan, D.L. Dill, E.A. Emerson, A partitioning methodology for BDD-based verification, in: International Conference on Formal Methods in Computer-Aided Design, 2004, pp. 399–413.
[43] R.D. Shachter, B. D'Ambrosio, B. Del Favero, Symbolic probabilistic inference in belief networks, in: International Conference on Artificial Intelligence, vol. 90, 1990, pp. 126–131.
[44] Y. Shen, A. Choi, A. Darwiche, Tractable operations for arithmetic circuits of probabilistic models, Adv. Neural Inf. Process. Syst. 29 (2016) 3936–3944.
[45] G. Torta, P. Torasso, On the role of modeling causal independence for system model compilation with OBDDs, AI Commun. 20 (1) (2007) 17–26.
[46] H. Zhao, M. Melibari, P. Poupart, On the relationship between sum-product networks and Bayesian networks, in: International Conference on Machine Learning, 2015, pp. 116–124.